VERSIONE PLUS RIVISTA+LIBRO+CD €14,90
VERSIONE STANDARD RIVISTA+CD €6,90
Periodicità mensile • MARZO 2004 • ANNO VIII, N.3 (78)
Poste Italiane • Spedizione in a.p. - 45% • art. 2 comma 20/b legge 662/96 - AUT. N. DCDC/033/01/CS/CAL

ioProgrammo

DATABASE A OGGETTI
UNA GUIDA PER PASSARE DAGLI RDBMS AGLI ORDBMS

Uno Sniffer fai da te

SPIARE I PC CONNESSI IN RETE

Scopriamo il codice che gli hacker usano per scovare password e monitorare la rete

- Uno sniffer a basso livello in Java
- Un progetto completo in VB
- Tutto il codice e le applicazioni nel CD

WEB SERVICES E VB 6
Impariamo a costruire e pubblicare un servizio con il Microsoft SOAP Toolkit

XML: IL CUORE DI OFFICE 2003
La suite diventa una piattaforma di sviluppo

UN INSTALLER IN JAVA
Realizzare un'alternativa ai pacchetti commerciali

SISTEMA
- L'accesso ai dati con ADO.NET
- Java: estrarre file da archivi Zip
- Realizziamo uno Scripting Engine
- Una calcolatrice in Delphi

INTERNET
- J2EE: un portale con il framework Struts
- Client di posta in Java: la gestione degli utenti

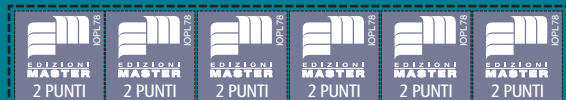
POCKETPC
- Estrarre file multimediali da SQLServer

ELETTRONICA
- Costruiamo un pannello di LED controllato da PC

ADVANCED
- Delegati ed eventi in C#
- Giochi: la ricerca della soluzione
- String matching: gli algoritmi

PROGRAMMARE AUTOCAD 2004 CON VBA

CONTENTS ▼

Questo mese su ioProgrammo

Anno VIII - n. 3 (78) Marzo 2004

▼ Brutti, sporchi e cattivi

Che musica ascoltate? Quali sono i film che preferite? Ultimo libro letto? Domande classiche e scontate, ma che una buona cena e un po' di vino rendono più urgenti. Credo che siamo tutti stanchi della classica iconografia che ritrae il programmatore-hacker con davanti, nell'ordine: un hamburger mangiucchiato, una coca mezza bevuta, musica in metallo nelle cuffie e collezione di calzini sotto il letto. Curiosi e vulcanici, i programmatori sono forse gli ultimi esponenti di una stirpe di uomini speciali: a metà fra ingegneri e artigiani, rientrano in quella stretta cerchia di persone capaci di rendere concrete le idee, di migliorare il lavoro e la vita delle persone.
Dimostrando una flessibilità sconosciuta ai più, i programmatori riescono a passare da un linguaggio all'altro, da un sistema all'altro, accrescendo la loro conoscenza in un continuo processo di crescita che rende stimolante e "avventurosa" buona parte delle loro giornate.
Ciononostante, ai programmatori sono spesso imposti ritmi insostenibili, con richieste che vanno da "offensive" banalità ai più complessi progetti.
Cosa ne direste di un bel sondaggio sulla nostra musica preferita, sui libri che leggiamo, sulle parole ed i pensieri che popolano la nostra fantasia? Ma forse è meglio lasciar perdere, meglio che ci vedano cattivi.

raffaele@edmaster.it

All'inizio di ogni articolo, troverete un nuovo simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre *\soft\codice* e *\soft\tools*) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.
Per scaricare software e codice da Internet, ogni mese indicheremo una password differente. Per il numero che avete fra le mani la combinazione è:

Username: **glory** *Password:* **box**

ioProgrammo

Anno VIII - N.ro 3 (78) - Marzo 2004 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale Massimo Sesti
Direttore Responsabile Romina Sesti
Responsabile Editoriale Gianmarco Bruni
Responsabile Marketing Antonio Meduri
Editor Gianfranco Forlino
Coordinamento redazionale Raffaele del Monaco
Redazione Antonio Pasqua, Thomas Zaffino
Collaboratori N. Alemanni, S. Ascheri, M. Autiero, L. Buono, M. Canducci, M. Casario, M. Del Gobbo, P. De Nictolis, F. Grimaldi, A. Ingegneri, A. Marroccelli, F. Mestrone, C. Pelliccia, P. Perrotta, M. Poponi, L. Spuntoni, E. Tavolaro, F. Vaccaro, I. Venuti, D. Visicchio
Segreteria di Redazione Veronica Longo

Realizzazione grafica Cromatika S.r.l.
Responsabile grafico: Paolo Cristiano
Coordinamento tecnico: Giancarlo Sicilia
Impaginazione elettronica: Aurelio Monaco

"Rispettare l'uomo e l'ambiente in cui esso vive e lavora è una parte di tutto ciò che facciamo e di ogni decisione che prendiamo per assicurare che le nostre operazioni siano basate sul continuo miglioramento delle performance ambientali e sulla prevenzione dell'inquinamento"

Certificato UNI EN ISO 14001
N. 9191 CRMT

Realizzazione Multimediale SET S.r.l.
Coordinamento Tecnico Piero Mannelli
Realizzazione CD-Rom Paolo Iacona

Pubblicità Master Advertising s.r.l.
Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite Daisy Zonato

Editore Edizioni Master S.r.l.
Sede di Milano: Via Cesare Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121206
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Amministratore Unico: Massimo Sesti

Abbonamento e arretrati
ITALIA: Abbonamento Annuale: ioProgrammo Basic(11 numeri): € 52,90 sconto 30% sul prezzo di copertina € 75,90.
ioProgrammo Libro (11 numeri + 6 libri): € 86,90 sconto 30% sul prezzo di copertina € 123,90. Offerte valide fino al 30/06/2004.
ESTERO: Abbonamento Annuale: ioProgrammo Basic (11 numeri): € 151,80. ioProgrammo Plus (11 numeri + 6 libri): € 257,00
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via Cesare Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- **cc/p n.16821878** o **vaglia postale** (inviando copia della ricevuta del versamento insieme alla richiesta);
- **assegno bancario non trasferibile** (da inviarsi in busta chiusa insieme alla richiesta);
- **carta di credito**, circuito VISA, CARTASI', MASTERCARD/EUROCARD, (inviando la Vs. autorizzazione, il numero della carta, la data di scadenza e la Vs. sottoscrizione insieme alla richiesta).
- **bonifico bancario** intestato a Edizioni Master S.r.l. c/o Banca Credem S.p.a. c/c 01 000 000 5000 ABI 03032 CAB 80880 CIN Q (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.
Sostituzioni: Inviare il CD-Rom difettoso in busta chiusa a: Edizioni Master Servizio Clienti - Via Cesari Correnti, 1 - 20123 Milano

Assistenza tecnica: ioprogrammo@edmaster.it

Servizio Abbonati:
tel.02 831212
@ e-mail: servizioabbonati@edmaster.it

Stampa: Rotoeffe Via Variante di Cancelleria, 2/6 - Ariccia (Roma)
Stampa CD-Rom: Deluxe Italy S.r.l. - via Rossini, 4 - Tribiano (MI)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Febbraio 2004

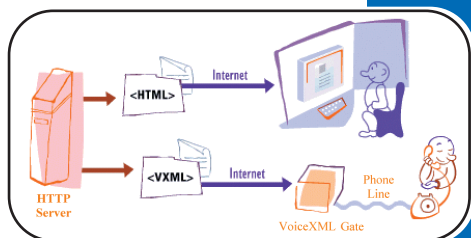
Edizioni Master edita:
Idea Web, Go!OnLine Internet Magazine, Win Magazine, PC Fun extreme, Quale Computer, DVD Magazine, Office Magazine, La mia Barca, ioProgrammo, Linux Magazine, Softline Software World, HC Guida all'Home Cinema, <tag/>, MPC, Discovery DVD, Computer Games Gold, inDVD, I Fantastici CD-Rom, PC VideoGuide, I Corsi di Win Magazine, I Filmissimi in DVD, La mia videoteca, Le Collection, Tv e Satellite.

IBM UNISCE ECLIPSE E DB2

Per la nuova versione del database di IBM sarà disponibile un plug-in che consentirà un semplice collegamento con l'ambiente open source Eclipse.
Stinger, è il nome scelto per la prossima versione di DB2, ed è dunque previsto un ampio supporto per lo sviluppo Java: oggetti come tabelle, indici e viste saranno gestibili direttamente dall'interno dell'IDE di Eclipse.
Oltre a fare da ponte fra le due piattaforme, il plug-in fornisce tool e wizard per semplificare tutta le gestione del database e consentire il completo controllo di DB2 a livello di codice.

www.eclipse.org

VOICEXML 2.0 AVANZA

La versione 2.0 di VoiceXML è già alla fase di "proposed recommendation" nell'ambito del W3C. Il che vuol dire che ha già passato lo scrutinio pubblico e sta per essere sottoposta agli ultimi esami prima di potersi fregiare della "W3C Recommendation".
VoiceXML 2.0 ha come obiettivo fondamentale quello di semplificare l'interazione fra applicazioni di case diverse, attraverso una massiccia standardizzazione degli elementi

che lo compongono.
VoiceXML 2.0 include una grammatica standard, un linguaggio di markup standard per la sintesi vocale ed una serie di formati audio standard.

www.voicexml.org

News

DA MICROSOFT, UN MILIARDO DI DOLLARI PER I PAESI POVERI

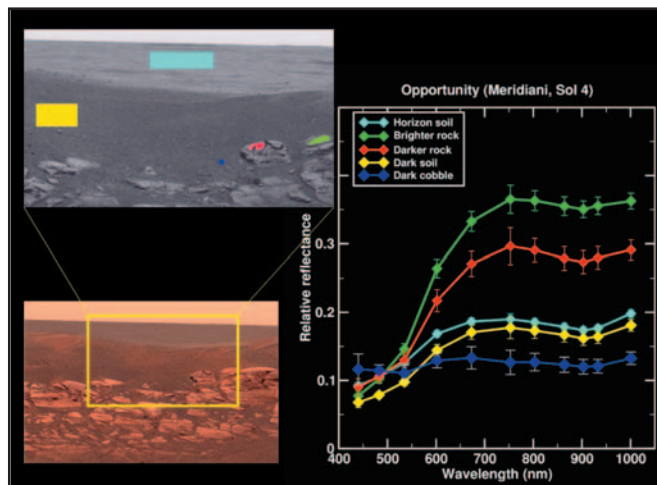
I paesi in via di sviluppo potranno beneficiare di questa somma attraverso l'Undp, l'ente americano per il Programma di Sviluppo delle Nazioni Unite.
L'ente collaborerà con Microsoft alla creazione di strutture nei paesi poveri, in cui le persone possano imparare l'uso del computer e migliorare le proprie prospettive di lavoro.
L'annuncio segue un progetto pilota condotto in Afghanistan, che ha consentito la creazione di 16 laboratori grazie ai quali saranno formate almeno 12.000 persone.

UNA MAIL FIRMATA GOOGLE?

Sono in molti ormai a credere che il più noto motore di ricerca stia per lanciare un nuovo servizio. La registrazione del dominio googlemail.com potrebbe essere dunque parte di una strategia che vedrebbe Google contrapporsi sempre più frontalmente al suo ex alleato Yahoo. Si aspettano ovviamente grandi cose dalla società che ha rivoluzionato il modo di interagire con Internet: quali potranno essere le novità di un servizio mail gratuito? La cosa più probabile è che in Google stiano lavorando ad un nuovo sistema per bloccare lo spam... ecco, se ci riuscissero, le menti Google porterebbero una nuova e attesa rivoluzione in Internet.

www.google.com

JAVA GUIDA SU MARTE

I recenti successi della NASA su Marte hanno diffuso un rinnovato entusiasmo per le esplorazioni spaziali.
Suggestive e bellissime, le foto che i robot dell'agenzia americana inviano da Marte sono state scaricate da milioni di internauti. È interessante scoprire che tutta l'infrastruttura informatica che governa i movimenti di Spirit (il robottino marziano) è stata creata in Java.
La NASA ha adottato questa piattaforma per la sua facilità e per il basso costo. Tutte le operazioni di Spirit sono guidate attraverso una plancia di comando che, grazie a Java, rende visuali tutte le operazioni più complesse. James Gosling (Vice Presidente di Sun) è direttamente coinvolto nel progetto ed ha sottolineato che all'inizio la NASA era stata attratta dal Java per le sue doti nella manipolazioni dei dati ma, quello che ha definitivamente convinto gli scienziati, è stata la possibilità di far dialogare i più disparati sistemi hardware. "Alla NASA trovate scienziati di tutto il mondo che parlano lingue diverse e

IL W3C APPROVA XML SCHEMA API

Lo scorso gennaio sono state approvate le specifiche *"XML Schema API"*, proposte da IBM e X-Hive. Il documento definisce una *Application Program Interface* che offre programmi e script dinamici per accedere al PSVI.
Quest'ultima specifica è stata introdotta e descritta nella sezione Normative Appendix C *(Outcome Tabulations)* della raccomandazione W3C *"XML Schema Part 1: Structures, C.2 Contributions to the Post-Schema-Validation Infoset"*.

www.w3.org

decidono assieme lo sviluppo della missione. Solo quando dialogano con il robot utilizzano lo stesso linguaggio: Java." Una delle applicazioni utilizzate alla NASA per dialogare con Spirit è "Maestro" che può essere scaricato da chiunque all'indirizzo *http://mars.telascience.org/*. Insieme all'applicazione, saranno scaricati anche una serie di dati raccolti direttamente su Marte. Grazie a questi dati il software è in grado di ricostruire lo scenario in cui Spirit agisce e voi potrete esplorare la zona con gli stessi strumenti utilizzati dai tecnici della NASA.
È anche possibile effettuare degli aggiornamenti sui dati, man mano che Spirit prosegue la sua esplorazione...

http://marsrovers.jpl.nasa.gov/

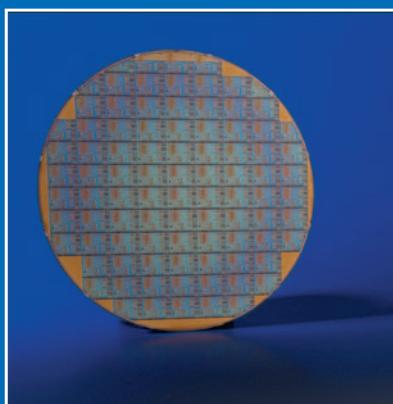
UN FONDO PER DIFENDERE LINUX

L'attacco che SCO ha lanciato contro Linux è fonte di preoccupazione per molti programmatori e, ancor più, per molte grandi aziende che hanno investito su questa piattaforma. In prima linea ci sono Ibm e Intel che hanno infatti aderito prontamente alla campagna lanciata dagli Open Source Development Labs (Osdl) per la costituzione di un fondo per difendere legalmente Linux. L'obiettivo è di raggiungere i dieci milioni di dollari e, grazie proprio alle donazioni di queste grandi aziende, il fondo ha subito toccato quota tre milioni.

www.osdl.org

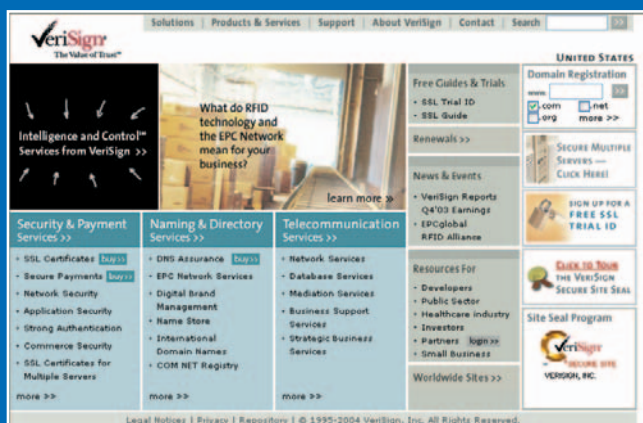
SONY INVESTE NEI CHIP

Oltre 325 milioni di dollari è la cospicua cifra che Sony investirà nell'impianto che IBM sta costruendo a New York per la produzione di chip di nuova generazione.
La produzione pilota dovrebbe cominciare entro la prima metà del 2005 e, a regime, in quegli stabilimenti saranno prodotti chip con dimensione del canale di 65 nanometri, dunque più potenti e piccoli. Si cercherà anche di abbattere i costi di produzione, con la realizzazione di wafer più grandi (300 millimetri).
Il nome in codice della nuova linea di processori è "cell'" e sono in molti a credere che rappresenteranno il cuore delle future console da gioco della casa giapponese, anche se la Sony ha più volte ribadito che vuol fare di "cell'" motore di qualsiasi dispositivo multimediale del prossimo futuro.

www.sony.com

MICROSOFT, XML E BREVETTI

In Nuova Zelanda e in Europa Microsoft ha richiesto la registrazione di un brevetto che, secondo alcuni, potrebbe impedire alle applicazioni di terze parti di aprire i documenti salvati con Office 2003. Il brevetto prevede di dotare i documenti Word 2003 di un formato XML nativo che non dia modo ai concorrenti, in primis OpenOffice e WordPerfect, di interpretare e manipolare i file creati con Office 2003. "Mentre lo standard XML è di per sé gratuito, nulla preclude ad un'azienda di brevettare una specifica implementazione software che includa elementi di XML", ha affermato il portavoce di Microsoft Mark Martin. "La presenza di questo brevetto in Nuova Zelanda (e Unione Europea, N.d.R.) non va a modificare in nessun modo l'impegno attraverso cui Microsoft ha aperto i propri schemi XML". Lo stesso Mark Martin ha affermato che non avrebbe senso per Microsoft porre un divieto allo sviluppo di uno standard, XML, che "ha attivamente contribuito a far nascere e promuovere".

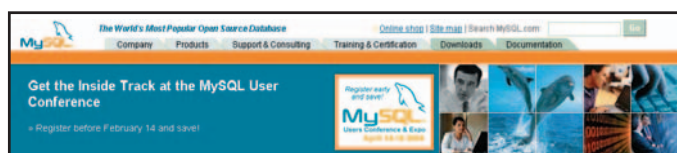
VERISIGN GESTIRÀ GLI RFID

La EPCglobal Inc, l'organizzazione che sta mettendo a punto gli standard per la tecnologia di identificazione a radiofrequenza, ha annunciato di aver scelto VeriSign per la gestione della directory principale contenente i codici di prodotto in standard RFID. VeriSign ha costruito la sua fama nel campo della sicurezza e dei servizi e, per conto di ICANN, già gestisce i registri .net e .com per Internet.

Electronic Product Code Network, attraverso la directory principale che sarà gestita da VeriSign, assegnerà codici distinti ai prodotti contrassegnati dai tag RFID basati sugli standard EPC. La rete EPC sarà offerta alle aziende sulla base di una sottoscrizione, il cui costo momento non è stato rivelato.

www.verisign.com

MYSQL E JBOSS: LA FORZA DELL'UNIONE

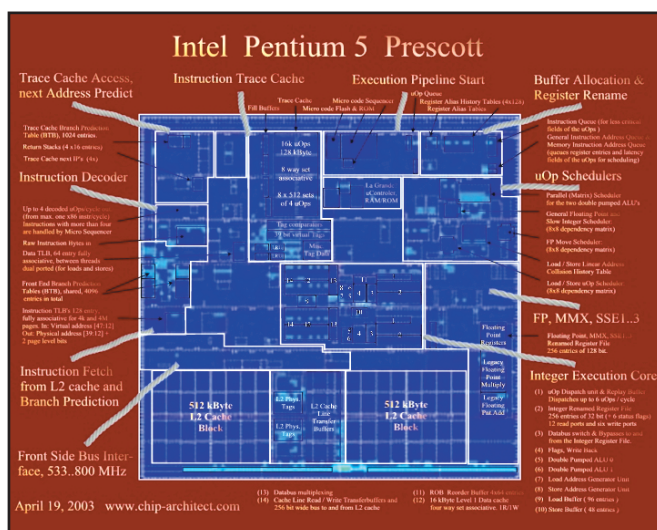
I due colossi dell'open source si sono alleati integrando il più diffuso database MySQL con l'ottimo l'application server Java Boss.
In questo modo risulterà semplificato lo sviluppo e il deployment delle applicazioni Web-based. JBoss semplificherà il processo di installazione del proprio application, integrando la famiglia di database open source MySQL. Anche il software MaxDB, sviluppato congiuntamente da MySQL e SAP, rientra in questo accordo che potrebbe dare un notevole scossone al mondo dello sviluppo.

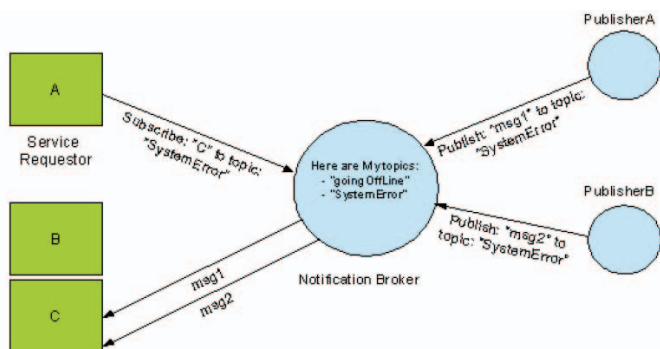
www.mysql.com

CONVERGENZA FRA WEB SERVICES E GRID COMPUTING

Un gruppo di aziende guidato da IBM ha proposti una serie di specifiche per la standardizzazione dell'utilizzo di Web Services come base per la costruzione di sistemi di Grid Computing.
Le specifiche sono sostanzialmente tre:

- **WS-Notification**, per la gestione di eventi e trigger;
- **WS-Resource Lifetime**, che consente agli utenti di specificare il periodo durante il quale una risorsa è valida
- **WS-Resource Properties**, che consente di definire le regole attraverso cui i dati associati ad una risorsa possono essere interrogati via Web Services.

INTEL LANCIA PRESCOTT

È pronta la prossima generazione di microprocessori: "Prescott". Intel ha effettuato il lancio ufficiale, senza però risolvere i dubbi sulla possibilità di lavorare a 64 bit. I "tagli" previsti per i primi esemplari sono 2.8GHz, 3.0 GHz e 3.2GHz.

Tra le principali differenze rispetto ai Pentium attualmente in commercio, si segnalano una cache di secondo livello raddoppiata (da 512 KB ad 1 MB), tredici nuove istruzioni, oltre al fatto di essere i primi chip di casa Intel ad essere prodotti con tecnologia a 90nm.

www.intel.com

Tutte queste specifiche fanno parte di un progetto più ampio, WS-Resource Framework, che ha l'obiettivo di rendere lo stack dei Web Services coerente con l'Open Grid Services Infrastructure. Il WS-Resource Framework è stato sottomesso ad OASIS per ottenere la standardizzazione: dai tre ai sei mesi saranno necessari per vedere le prime implementazioni reali.

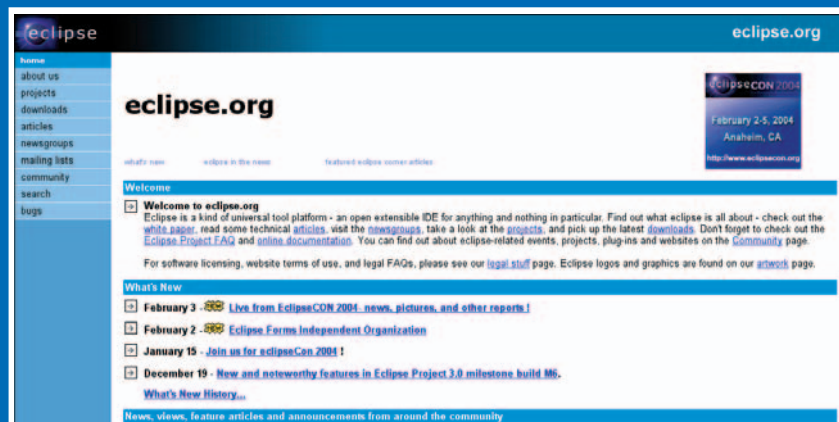
www.ibm.com/developerworks/grid/

UN SOFTWARE CHE RICONOSCE I PROPRI ERRORI

I ricercatori del MIT sono al lavoro su nuovi algoritmi che possano permettere ai sistemi software di rilevare i propri malfunzionamenti e riparare gli errori al volo senza dover tornare ad uno stadio precedente. Lo scopo è realizzare sistemi che possano restituire dati corretti subito dopo un errore, riconoscendo con grande velocità un eventuale difformità.
La strategia alla base di questa tecnica consiste nell'esaminare una modello astratto delle strutture dati costruito in modo da evidenziare eventuali incongruenze con la stessa semplicità con cui una radiografia mostra una frattura ossea. Quando il sistema incontra una inconsistenza, mette in moto una serie di azioni per risolvere il problema.

ECLIPSE ORA CAMMINA CON LE SUE GAMBE

Da oggi in poi il progetto Eclipse camminerà con le sue gambe, diventando un consorzio indipendente cui potranno aderire tutte le aziende che lo vorranno. Il modello è quello di altre realtà open source, come l'Apache Foundation e sono inviatati a partecipare tutti i soggetti che sono interessati: università e organizzazioni no-profit non dovranno versare alcuna quota di partecipazione, mentre per le società commerciali è prevista un quota associativa pari a $5.000. Il ridimensionamento del ruolo di IBM potrebbe portare ad un riavvicinamento di SUN che ha infatti reso pubblica una lettera aperta in cui si congratulava con il consorzio per la scelta di libertà effettuata. Proprio questa lettera è una testimonianza dell'interesse che SUN ancora nutre in Eclipse. Aspettiamo tutti con speranza qualche passo importante: l'attuale divisione nel mondo Java rischia infatti di danneggiare in modo grave una comunità di sviluppatori che merita ben altre attenzioni.

www.eclipse.org

FIOCCO ROSA PER LA JAVA TOOLS COMMUNITY

Con un caldo benvenuto da parte della comunità degli sviluppatori, nasce questa iniziativa finalizzata a facilitare la diffusione di standard Java "tool friendly".
La Jtc promuoverà la creazione e l'adozione delle JavaSpecification Requests (Jsr) che favoriscano la creazione di tool integrabili attraverso l'adozione di uno standard comune già in fase di progettazione.
Si introduce, spiega un comunicato, un criterio di misura di quanto sia semplice costruire tool di sviluppo capaci di supportare un particolare standard o una specifica tecnologia (criterio di "toolability").
Come risultato, gli sviluppatori avranno la possibilità di utilizzare più facilmente la tecnologia Java per creare nuove applicazioni e renderne più veloce la diffusione.
Tra i partner fondatori della Jtc, figurano produttori di software come Bea Systems, Compuware, Embarcadero Technologies, Iopsis Software, Jetbrains, Oracle, Quest Software, Sap, Sas e Sun Microsystems.

www.javatools.org

SOFTWARE SUL CD

Una selezione dei migliori tool di sviluppo proposta dalla redazione di ioProgrammo

Borland JBuilder X Foundation

Potente e gratuito, uno dei migliori e più completi ambienti di sviluppo per Java.

Un ambiente di sviluppo integrato che consente di velocizzare e rendere più piacevole sia la stesura del codice che il debugging. Ricco di wizard e di progetti dimostrativi, JBuilder può essere utilizzato subito in modo proficuo da tutti gli sviluppatori. JBuilder X Foundation va a prendere il posto della Personal Edition anche se, a differenza della vecchia edizione, consente anche lo sviluppo di applicazioni commerciali. La edizione Foundation include l'editor, un debugger ma non offre il supporto per Struts, JSP e per le funzioni più avanzate di J2EE che caratterizzano la edizione full.

L'INTERFACCIA

Utilizzando JBuilder, salta subito all'occhio la maggiore flessibilità dell'interfaccia che consente di personalizzare in pochi istanti l'ambiente di lavoro, ad esempio liberandolo dalle barre che non ci interessa utilizzare. Molte migliorie sono state apportate a quello che per lo sviluppatore è il pane quotidiano: la scrittura del codice. In particolare, ci piace segnalare la funzione che cambia in grigio il colore delle variabili che non sono più utilizzate, assieme ad una utile sezione "to-do" in cui è possibile segnare le cose da fare. Mentre scriviamo, il codice è tenuto sotto controllo da un parser che ne verifica la correttezza. Nei progetti di grandi dimensioni, questa funzione di indubbio vantaggio può costituire un problema in ordine alla lentezza con cui il parser effettua il suo lavoro. Per ovviare a questo inconveniente è possibile impostare il ritardo con cui il parser comincia ad agire dopo la digitazione: un clic con il tasto destro sul pannello structure, scegliemo la voce Properties, e dalla finestra di pop-up impostiamo il ritardo in millisecondi.

Fig. 1: *L'interfaccia si presenta ricca e ben organizzata, sarebbe difficile chiedere di più!*

PERSONALIZZARE

Notevoli sono le personalizzazioni consentite nell'editor: ad esempio, è possibile cambiare dimensione e font dei caratteri utilizzati ed è possibile abilitare e disabilitare la visualizzazione dei numeri di linea. È possibile scegliere i colori e quali pannelli presentare e, all'interno di ogni pannello, specificare ulteriori personalizzazioni. Per accedere alle personalizzazioni dell'editor, dal menu *Tools*, scegliere la voce *Prefernces...*, e dal pop-up indicare la voce Editor.

CODICE AUTOMATICO

JBuilder fornisce numerosi tool per velocizzare lo sviluppo delle applicazioni:

Fig. 2: ***È possibile personalizzare massicciamente l'ambiente di lavoro.***

Fig. 3: ***La finestra di Design. Si distinguono quattro elementi fondamentali: l'area di lavoro, la palette di componenti, l'inspector con le proprietà e la structure panel.***

creazione visuale delle interfacce, Wizard per la generazione automatica di codice, generazione automatica di JAR avviabili, oltre al supporto dei template come ausilio alla scrittura di codice corretto.
Particolarmente rilevanti gli strumenti che abbiamo a disposizione per la generazione di interfacce: oltre alle numerose palette di oggetti pronti per essere trascinati nell'area di lavoro, troviamo un completo inspector che consente di visualizzare e modificare tutte le proprietà di qualsiasi oggetto. Anche la structure pane contribuisce a semplificare la creazione dell'interfaccia, mostrando in sempre molto chiaramente la gerarchia esistente fra gli elementi presenti.

PARTIRE DA MAGHI

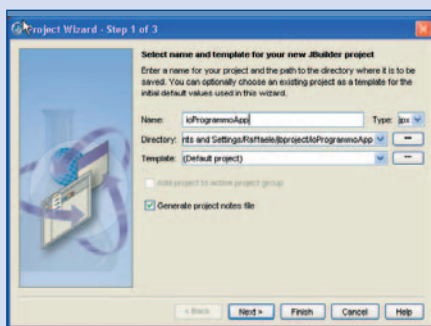
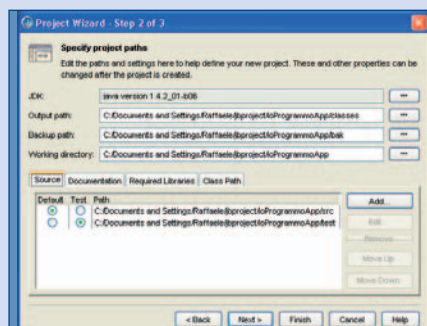
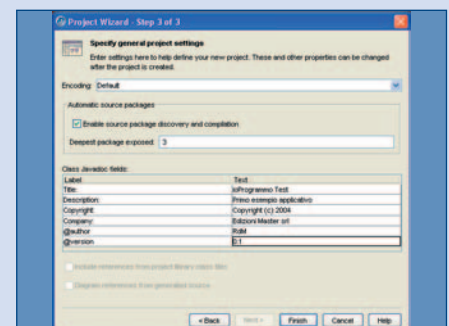
Abbiamo già detto che i wizard sono uno degli elementi chiave scelti da Borland per semplificare e rendere più rapida la scrittura di codice. Si può subito cominciare a godere dei benefici derivanti da questa tecnica scegliendo la voce New dal menu File. Si aprirà la cosiddetta Object Gallery. Una volta scelto il tipo di applicazione o componente che vogliamo realizzare che ci consentirà, avremo a nostra disposizione, già pronte, una serie di classi base che faranno da scheletro alla nostra applicazione reale. Sulla sinistra è possibile scorrere un piccolo menu ad albero in cui sono rappresentate le varie categorie cui appartengono gli oggetti realizzabili tramite wizard. Noterete molti elementi non selezionabili.
Alcuni sono disponibili solo in alcune circostanze, ad esempio dopo che sia stato aperto o creato un progetto.
Altre voci sono purtroppo deselezionate "a vita" perchè disponibili solo nelle versioni a pagamento di JBuilderX: Enterprise e Developer.

Fig. 4: ***Object Gallery: numerosi wizard ci guidano alla costruzione degli elementi fondamentali di un'applicazione.***

INSTALLAZIONE

Per una corretta installazione è necessario collegarsi al link *http://www.borland.com/products/downloads/download_jbuilder.html#* e cliccare su *"Enterprise Trial & Foundation"*, lasciare i propri dati evitando di scaricare il pacchetto di installazione. In pochi istanti riceveremo nella casella di posta elettronica la chiave di attivazione.

JBuilder X Foundation
Produttore: Borland
Sul web: www.borland.it
Prezzo: Gratuito
Nel CD: \JBuilderX

IL NOSTRO PRIMO PROGETTO

1 **Per attivare il Wizard, dal menu File, selezionare *New project*. Indicare il nome del nuovo progetto e spuntare la *casella Generate project notes file*. Un clic su *Next***

2 **Possiamo accettare tutti i dati proposti in questa seconda schermata e cliccare subito su *Next*. Andate comunque a verificare dove saranno salvate le classi del progetto.**

3 **Andiamo a specificare i vari campi con i nostri dati e clicchiamo su *Finish*. Saranno generati due file: una classe ed un file HTML di documentazione. Il progetto è pronto per essere sviluppato!**

ObJect Relational Bridge

Tool di mapping di Apache per la persistenza di oggetti Java

di David Visicchio

La Apache Software Foundation, ormai famosa per diversi progetti open source, tra i quali, tanto per citarne uno, il noto ed omonimo Web Server, ha messo a disposizione questo innovativo tool destinato agli sviluppatori: un sottoprogetto, creato all'interno del più ampio progetto DB, volto al supporto di soluzioni open source relative all'uso di database.
ObJectRelationalBridge rappresenta la soluzione al problema della persistenza di oggetti di un'applicazione Java su una tradizionale base dati di tipo relazionale. Proprio per questo motivo, il nome stesso indica un "ponte" tra il mondo ad oggetti, tipico di Java, e quello relazionale degli RDBMS (Relational database management system). La persistenza di oggetti su database relazionali presenta, come inconveniente, la necessità di definire un mapping tra il modello delle classi e quello relazionale. Ciò può essere agevolato da tool appositi che permettono di definire le regole di mapping e che agiscono come interfaccia tra l'applicazione e il database. Il mapping, infatti, è basato su un file XML e definito in un repository. In esso sono memorizzati i metadata, che servono all'esecuzione della "traduzione" da classi a tabelle.
Una volta definito, il mapping è dinamico e può essere modificato a run-time.
Inoltre sono supportate le possibili associazioni 1:1, 1:n e m:n tra classi.

PIÙ DATABASE

ObJectRelationalBridge consente di utilizzare differenti database (compatibili JDBC) anche contemporaneamente, partizionando così la

Fig. 1: L'home page del progetto.

CREARE I METADATA PER IL MAPPING

```
package org.apache.ojb.tutorials;

public class Product
{
    /** price per item*/
    private Double price;

    /** stock of currently available items*/
    private Integer stock;

    /** product name*/
    private String name;

    /* Getters and Setters not shown for tutorial

    /** Artificial primary-key */
    private Integer id;
}
```

1 **Partiamo da una semplice classe priva di relazioni con altre classi e composta da tre attributi (price, stock e name). Introduciamo un attributo fittizio che sarà usato nel database per identificare l'oggetto.**

```
CREATE TABLE product
(
    id INTEGER PRIMARY KEY,
    name VARCHAR(100),
    stock INTEGER,
    price DOUBLE
)
```

2 **In questo semplice caso, usiamo, per mappare la classe *Product*, una sola tabella in cui ogni riga corrisponderà ad un oggetto della classe. Scriviamo quindi il codice SQL relativo alla creazione della tabella.**

```
<class-descriptor
    class="org.apache.ojb.tutorials.Product"
    table="PRODUCT"
>
  <field-descriptor
     name="id"
     column="ID"
     primarykey="true"
     autoincrement="true"
  />
  <field-descriptor
     name="name"
     column="NAME"
  />
  <field-descriptor
     name="price"
     column="PRICE"
  />
  <field-descriptor
     name="stock"
     column="STOCK"
  />
</class-descriptor>
```

3 **Infine definiamo nel file *repository.xml* le regole di mapping tra gli attributi della classe e le colonne della tabella.**

persistenza degli oggetti, rendendoli trasparenti all'applicazione mediante un'unica interfaccia.
Sono supportate diverse API per la persistenza quali:

- *ODMG 3.0*
- *JDO*
- *Persistence Broker*

L'ODMG è uno standard che definisce le API da utilizzare per la persistenza degli oggetti.
L'Object Data Management Group è il nome di un gruppo, fondato nel 1991, costituito dalle maggiori società di informatica con il compito di definire gli standard nel campo degli ODBMS (Object database management system) ed in generale nella persistenza degli oggetti. Nel 2001 il gruppo è stato sciolto, con il rilascio definitivo dell'ultimo standard ODMG 3.0. Viceversa il JDO (Java Data Objects) è il nuovo standard di interfaccia, per il linguaggio Java, verso i sistemi di gestione dei dati. È stato sviluppato dalla Java Community Process con il contributo dei maggiori produttori di database e tecnologie persistenti. Il JDO permette agli sviluppatori di rendere direttamente persistenti oggetti Java in differenti data store (relazionali, ad oggetti, xml, ecc.). Nella versione attuale di ObJectRelationalBridge purtroppo l'implementazione JDO non è completata al 100% (essa è prevista con la release 2.0 del tool).
Il PersistenceBroker rappresenta il kernel del tool e le relative API sono piuttosto di basso livello.

STRUTTURA

La struttura di *ObJectRelationalBridge* è mostrata in Fig. 2.

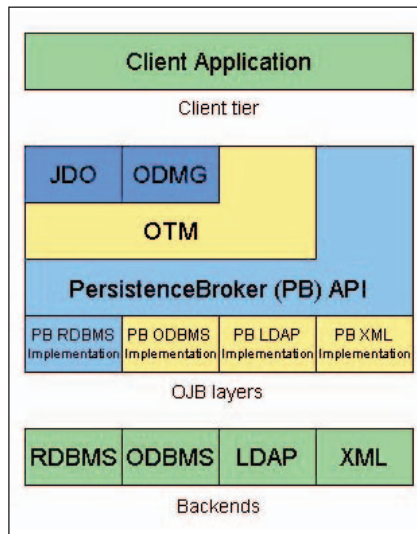



Fig. 2: ***I componenti principali che costituiscono OBJectRelationalBridge.***

Lo strato a livello più basso si basa sul PersistenceBroker, che offre un'architettura scalabile e multiserver. Al di sopra è presente un Object Transaction Manager (OTM) che comprende una serie di funzionalità comuni sia alla parte JDO sia a quella ODMG, quali, ad esempio, la gestione dei lock, il ciclo di vita degli oggetti, ecc. Infine ci sono i moduli relativi all'implementazione delle specifiche JDO e ODMG.

CONCLUSIONI

Tra le caratteristiche più avanzate, va menzionato soprattutto il supporto sia del pessimistic sia del optimistic locking. In tal modo si possono impostare modalità diverse di lock sugli oggetti in base al carico previsto dell'applicazione. In definitiva possiamo concludere che si tratta di un tool senz'altro prezioso in fase di sviluppo di un'applicazione Java.
Sicuramente l'implementazione completa del JDO, e la maggiore diffusione di questo standard nel futuro, permetterà una migliore utilizzabilità.
Al momento se ne consiglia l'uso mediante l'interfaccia ODMG, un tentativo di standardizzazione non sempre pienamente rispettato dal mercato.

☑ ObJectRelational Bridge

Produttore: Apache Software Group
Sul web: http://db.apache.org/ojb/
Prezzo: Gratuito (Open Source)
Nel CD: \OJB

ACCESSO TRAMITE ODMG

```
Implementation odmg = OJB.getInstance();
Database db = odmg.newDatabase();
db.open("default", Database.OPEN_READ_WRITE);

/* ... use the database ... */

db.close();
```

1 **Vediamo come utilizzare la classe *Product*. La nostra applicazione deve per prima cosa aprire una connessione al database e chiuderla dopo aver effettuato le operazioni desiderate.**

```
public static void storeProduct(Product product)
{
    Implementation impl = OJB.getInstance();
    Transaction tx = impl.newTransaction();
    tx.begin();
    tx.lock(product, Transaction.WRITE);
    tx.commit();
}
```

2 **Rendiamo persistente un oggetto *Product*. Il metodo *OBJ.getInstance* ritorna un'istanza dell'implementazione ODMG con cui apriamo una transazione. Dopo aver impostato un lock in scrittura sull'oggetto effettuiamo il commit.**

```
public static void sellProduct(Product product, int number)
{
    Implementation impl = OJB.getInstance();
    Transaction tx = impl.newTransaction();
    tx.begin();

    tx.lock(product, Transaction.WRITE);
    product.setStock(new Integer(product.getStock().intValue() -  number));

    tx.commit();
}
```

3 **L'aggiornamento di un oggetto persistente avviene sempre all'interno di una transazione. Il codice ottiene un lock in scrittura, modifica l'oggetto ed effettua il commit.**

JewelBox Builder

Una valanga di prodotti Open Source, per avere una completa piattaforma di sviluppo Java pronta ad essere utilizzata.

Uno sviluppatore può utilizzare proficuamente moltissimi tool gratuiti ed opern source. Spesso però l'estrema abbondanza di questi oggetti può mettere in crisi lo sviluppatore e portarlo a fare scelte sbagliate o improprie. IDE, database, web server, application server: per ognuna di queste categorie, l'offerta è abbondante e risulta spesso difficile prendere una decisione.

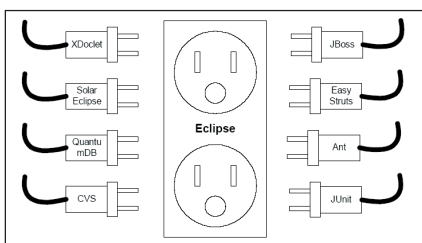



Fig. 1: *È sicuramente vantaggioso avere in un unico pacchetto Eclipse insieme ai suoi migliori plug-in.*

JewelBox Builder rappresenta proprio una risposta a questa esigenza di chiarezza. In questo pacchetto trovate una selezione dei migliori strumenti gratuiti attualmente disponibili, tutti armonizzati in modo

Fig. 2: *Servizi e Framework inclusi nel pacchetto.*

Fig. 3: *Tra i tools, trovate i migliori prodotti attualmente disponibili come Open Source.*

da poter lavorare assieme proficuamente:

Java 2 SDK SE 1.4.2
L'indispensabile piattaforma per chiunque voglia sviluppare in Java.

L'ambiente di sviluppo Sun che negli ultimi anni si è imposta come la prima scelta per i programmatori che lavorano in ambito multipiattaforma. In questa versione, nuove funzionalità e migliori prestazioni arricchiscono l'ambiente.

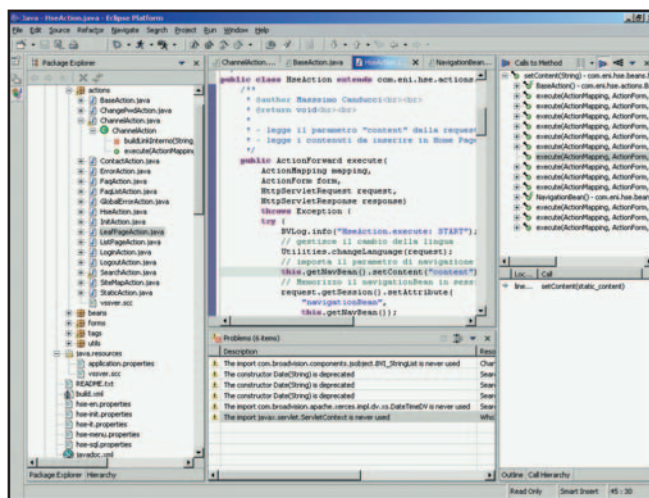



Fig. 4: *L'interfaccia di Eclipse.*

Rich client application e *Web Services* sono i campi in cui sono più evidenti i miglioramenti. Tra i miglioramenti che più faranno gola agli sviluppatori ci sono sicuramente quelli inerenti *Swing*. Davvero ghiotti i due nuovi *look&feel:* GTK+ e, finalmente, Windows XP. Anche le applicazioni Java possono così integrarsi pienamente nell'ambiente visuale del più recente Windows. Anche GTK+ risulta molto interessante: attraverso un semplice resource file, è possibile settare i parametri fondamentali del *look&feel*. Sempre in ambito Swing, è da notare la pesante cura dimagrante cui è stata sottoposta *JFileChooser*, che risulta essere ora molto più veloce della precedente versione, in alcuni casi anche 300 volte più veloce!

Eclipse 2.1.1
Un IDE ampiamente personalizzabile per la realizzazione di progetti Java di qualsiasi dimensione

Eclipse è una piattaforma integrata per lo sviluppo, il debug ed il test di applicazioni enterprise la cui architettura di base, dal valore iniziale di circa 40 milioni di dollari, è stata donata da IBM alla comunità Open Source. Il senso di *Eclipse* è quello di fornire agli sviluppatori degli strumenti di sviluppo sempre più sofisticati e potenti grazie ad un unico framework che integri potenzialmente tutti i linguaggi, gli Ide, i tool di modeling e gli strumenti di collaborazione esistenti. Si tratta di un framework basato su un sistema di API pubbliche e di plug-in le cui specifiche di sviluppo sono di pubblico dominio e questo permette a chiunque di sviluppare i propri

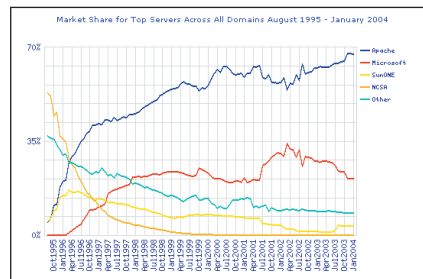



Fig. 5: *Su una base di oltre quaranta milioni di domini, le statistiche fornite da Netcraft fino al gennaio 2004 sono inequivocabili.*

plugin e renderli disponibili alla comunità Open Source oppure crearne versioni particolari a pagamento.

Apache 2.0.47
Il più diffuso Web Server al mondo

Questo programma non ha certo bisogno di presentazione, in poche parole è il server web più utilizzato al mondo: più del 56% dei server web utilizzano Apache. I motivi del successo di questo server sono da ricercarsi nella corretta implementazione dei protocolli HTTP, nella robustezza, nella sicurezza e nella disponibilità per qualsiasi piattaforma hardware/software. Attualmente esistono due “linee di produzione” la 1.3 e la più moderna 2.0, per maggiori informazioni *http://httpd.apache.org/*

ArgoUML
Un potente ambiente per la costruzione di diagrammi UML

Si tratta del primo storico prodotto Open Source per la progettazione visuale UML realizzato presso l'Università della California, il cui scopo è fornire un valido supporto all'analisi e alla progettazione di sistemi software Object Oriented.

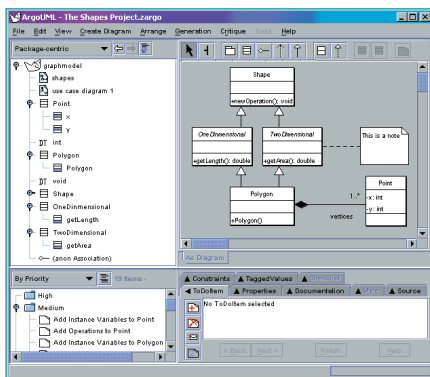



Fig. 6: Uno screenshot di Argo UML.

È un progetto che coinvolge circa un centinaio di persone tra ricercatori e studenti. *ArgoUML* è conforme alle specifiche UML 1.3 ed è l'unico case tool (a parte Poseidon) che implementa il metamodello UML esattamente come è specificato.

JBoss 3.2.2
L'application server dei professionisti

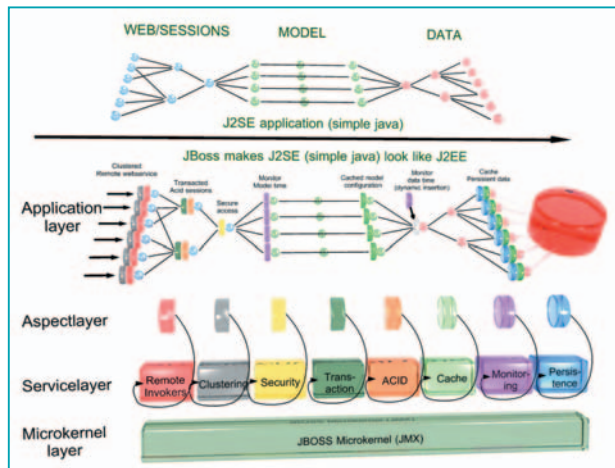



Fig. 7: Il successo di JBoss è legato alla sue doti di robustezza e semplicità

JBoss è un application server Open Source scritto interamente in Java. Grazie a *JBoss* è possibile far girare componenti EJB *(Enterprise Java Beans)* e qualsiasi applicazione sviluppata attraverso la tecnologia J2EE di Sun. Grazie ai connettori JMX, *JBoss* consente una notevole flessibilità e si presta agli utilizzi più disparati. L'architettura modulare di JBoss poggia su JMX *(JBoss Microkernel),* un framework che si fa carico esporre le interfacce applicative sia ad attori remoti, sia a oggetti locali. In particolare è possibile sviluppare un proprio transaction-manager ed un proprio persistence-manager. Tutta l'architettura è concepita in maniera modulare e costruita a layer indipendenti, cosa che ha consentito uno sviluppo costante delle funzionalità e della stabilità della piattaforma.

MySQL
Il migliore è più diffuso database Open Source

Velocissimo e affidabilissimo, trova nell'accoppiata con PHP la migliore combinazione. Tra le caratteristiche che ne hanno fatto il preferito per milioni di utenti vi è la capacità di gestire con disinvoltura grandi quantità di dati: centinaia di migliaia, milioni di record non sono un problema! Molte mailing list della grandezza di diversi Gigabyte sono ospitati proprio su *MySQL*.
Altro punto forte è la grande stabilità che non vacilla nemmeno di fronte a centinaia di accessi concorrenti, e questo dato ha fatto di *MySql* il preferito per le applicazioni Web.

LA SCELTA

Le versioni incluse nel JewelBox Builder non sempre le più recenti. Questo corrisponde ad una precisa scelta di 10x Software, l'azienda che ha deciso di distribuire il pacchetto.
L'obiettivo era infatti quello di fornire un insieme il più possibile stabile e omogeneo, invece di privilegiare componenti più aggiornati ma che ancora dovessero dimostrare la loro effettiva stabilità.

Apache Ant
Del tutto simile a Make, interviene nella fase di build di un'applicazione

CVSNT
Un tool per il versioning delle applicazioni

SolarEclipse
Un plug-in per Eclipse che consente la evidenziazione sintattica di HTML, JSP e XML

EasyStruts
Plug-in per Eclipse che semplifica la costruzione di applicazioni basate su framework Struts

JUnit
Un framework per il testing su Java

Quantum DB
Plug-in per Eclipse che permette agli sviluppatori di interfacciarsi con un database relazionale.

Nutrita e ottimamente scelta la lista del plug-in.

☑ JewelBox Builder
Produttore: 10x Software
Sul Web: www.10xsoftware.com
Prezzo: Gratuito
Nel CD: \JewelBox

XMLSpy 2004 Professional Edition R3

Il massimo per lo sviluppo XML

XMLSPY è ormai lo standard de facto nel mondo dello sviluppo XML professionale. *XMLSPY Pro* agevola la costruzione di applicazioni XML-based, siti Web, Web Services e tutto quanto ruota attorno all'XML, la stella incontrastata dell'attuale panorama della programmazione. È ovviamente possibile validare documenti XML sulla base di DTD ed è possibile trasformare i documenti utilizzando regole XSL, il tutto attraverso un apposito editor DTD, un editor grafico per *XML Schema* ed un processore XSLT.

INTERFACCIA

XMLSPY 2004 dispone di diverse finestre per mostrare i vari aspetti dei documenti XML che vogliamo elaborare. Le finestre sono divise in tre aree verticali: alla sinistra si notano le finestre *Project* e *Info*, nell'area centrale è possibile scegliere le diverse viste *(Grid view, Schema/WSDL design view, Text view, Authentic view, Browser view)*, alla destra troviamo infine tre diversi pannelli per inserire e modificare rispettivamente *Elementi, Attributi* ed *Entities.*

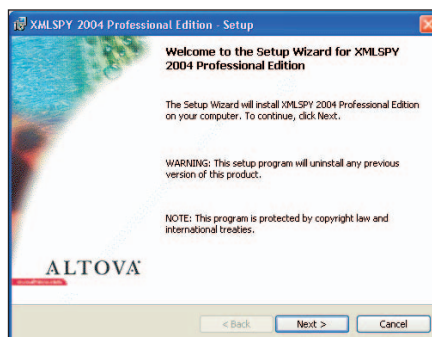



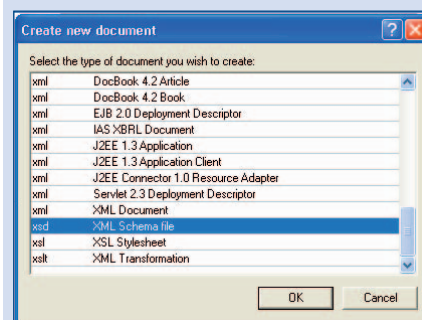
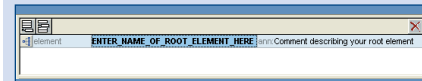
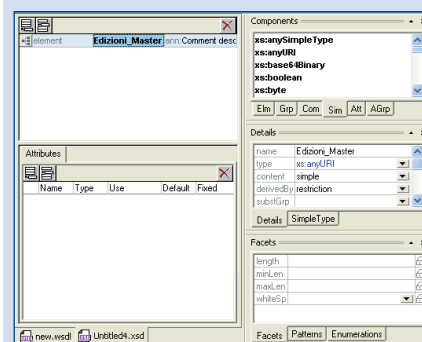
Fig. 2: *L'installazione è semplicissima e non richiede alcun intervento.*

INSTALLAZIONE

Già nelle fasi iniziali dell'installazione è possibile decidere il livello di integrazione con Windows, scegliendo se associare alcuni file a XMLSPY e se integrare la voce relativa a XMLSPY nei menu contestuali di *Explorer.* Alla fine dell'installazione è possibile scaricare automaticamente dei alcuni tool aggiuntivi. Al primo avvio ci viene richiesto il nostro nome e un indirizzo mail cui verrà spedita in pochi, istanti, la chiave di attivazione necessaria ad attivare il software per quindici giorni.

Fig. 1: *Davvero completissima la versione Professional di XMLSpy.*

☑ XMLSpy 2004 Professional Edition R3
Produttore: Altova
Sul Web: www.xmlspy.com
Prezzo: $ 399.00
Nel CD: XMLSPYProfComplete2004.exe

CREARE UN NUOVO FILE SCHEMA

1 Cliccare sulla voce New dal menu File e selezionare nel dialog che appare la voce .xsd W3C XML

2 Apparirà uno schema vuoto e ci verrà chiesto di indicare il nome dell'elemento Root

3 A questo punto saremo pronti a popolare il nuovo documento

Revolution 2.1.2

Per creare applicazioni multipiattaforma

Per gli sviluppatori che vogliono creare applicazioni che possano girare su qualsiasi sistema operativo da Mac OS a Windows, passando per il sempre più popolare Linux: ecco pronto *Transcript*, un linguaggio di alto livello, integrato in Revolution, e che consente di sviluppare applicazioni efficienti in un tempo brevissimo rispetto a linguaggi più classici come Java, C++ o VB. Due caratteristiche su tutte: multipiattaforma, e brevità. Funzioni che prima richiedevano complesse subroutine, si risolveranno con una singola riga di codice: sintesti vocale, manipolazione acquisizione e manipolazione di filmati, interazione grafica con database, tutto integrato all'interno di un'architettura particolarmente user-friendly che contempla un largo uso del drag-and-drop. In questa versione è stato integrato il supporto per SOAP ed una libreria specifica per il parsing XML. Potete provare la versione che trovate nel CD per un massimo di trenta giorni, con una limitazione sul numero di righe di codice associabile ad ogni oggetto. A parte questa restrizione, troverete tutte le funzioni disponibili nella versione a pagamento e potrete anche distribuire le applicazioni sviluppate con questo pacchetto. Al primo avvio vi verrà chiesto se volete lanciare il programma in versione trial, effettuata questa scelta vi sarà proposto di collegarvi all'indirizzo *www.runrev.com/revolution/30 day-trial.html.* Lasciando il vostro indirizzo e-mail, in pochi istanti vi sarà inviato un codice di attivazione che potrete copiare e incollare nell'apposita maschera che si presenta all'avvio di Revolution.

Fig. 1: *Numerosi tool visuali aiutano il processo di sviluppo.*

☑ Revolution 2.1.2
Produttore: Runtime Revolution Ltd.
Sul Web: www.runrev.com
Prezzo: $399
Nel CD: revsetup.exe

Hackman Hex Editor 7.05 Light

Editor esadecimale e disassemblatore in un unico software

A dispetto del nome, questo prodotto non è dedicato esclusivamente al mondo hacker, ma a chiunque sia curioso di indagare la struttura dei programmi eseguibili. HackMan può infatti riportare in assembler qualsiasi eseguibile adatto ai Pentium. A dispetto della sua gratuità, uno dei migliori disassemblatori reperibili al momento. HackMan è anche un ottimo editor esadecimale e offre la capacità di criptare e decrittare file con un algoritmo a 128 bit. Tra le caratteristiche più interessanti annoveriamo il disk editor, la toolbar configurabile, un'interfaccia più veloce, il supporto per disassemblare codice Pentium 4 e molto altro ancora. Con la semplicità di un Word Processor, sarà possibile leggere e modificare qualsiasi file, disco e qualsiasi zona della RAM. Molto utile la possibilità di controllare qualsiasi azione direttamente da un'apposita riga di comando, oltre che attraverso l'interfaccia grafica. E' possibile arricchire di nuove funzionalità l'ambiente, grazie al supporto per plug-in che possono essere realizzati con l'apposito SDK presente nel pacchetto di installazione. Versione light, risultano disabilitate numerose funzioni. La licenza per la versione full ha un costo di $25,00

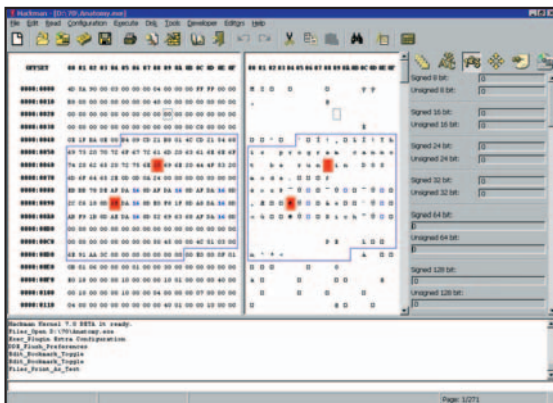
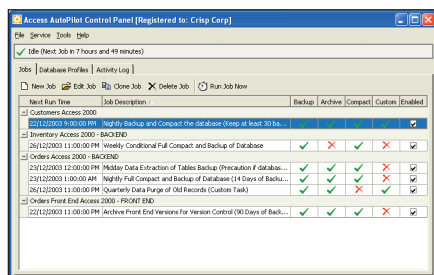



Fig. 1: *L'interfaccia è molto completa. Le prime volte può disorientare.*

☑ Hackman Hex Editor 7.05 Light
Produttore: TechnoLogismiki Athens
Sul Web: www.technologismiki.com
Prezzo: gratuito
Nel CD: hack705.zip

Access Autopilot 1.1.12

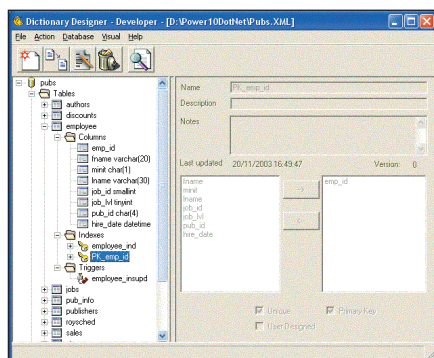
Un pilota automatico per Access

Ore e ore di lavoro risparmiate, grazie a questa applicazione che consente di automatizzare tutti i più frequenti (e noiosi) compiti che spettano ad un amministratore di database.
Operazioni di backup, compact periodico, e azioni di archivio, possono essere tranquillamente schedulate e fatte eseguire, ad esempio, durante la notte.

Autopilot gira come un servizio di Windows e, tra le funzioni più interessanti, consente di rilevare gli utenti connessi e di inviargli automaticamente delle notifiche personalizzate dall'amministratore. Versione di prova valida trenta giorni, il costo della versione completa è di 149.00.

AccessAutoPilot.msi

Advanced Data Dictionary Architect 1.0

Un valido aiuto per applicazioni database-driven

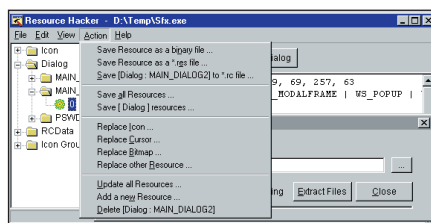
Un toolkit che aiuto lo sviluppatore in tutte le fasi di produzione del software e che consente di tenere sempre aggiornato il database-layer a business-layer. Le principali funzioni comprendono: la possibilità tenere traccia delle modifiche più recenti alla base di dati; la capacità di fare da supporto a processi di ingegneria del software con una visualizzazione del

database CASE-like (Computer Aided Software Engineering); creazione automatizzata di interi database; esteso supporto per XML; reverse-engineering di database già esistenti. In definitiva, Advanced Data Dictionary Architect, oltre a organizzare i meta-dati correlati ai database, consente di eseguire in modo automatico una serie di lunghi e noiosi procedimenti che caratterizzano la manutenzione e l'upgrading dei database. Lavora con SQL Server 7 e necessita che sia installato il Microsoft NET Framework 1.1. Versione dimostrativa, la versione completa ha un costo pari a 299,00.

SetupADDADemo.exe

Resource Hacker 3.4

Per "rubare" le risorse alle applicazioni

Una utility completamente gratuita che consente di reperire ed estrarre le risorse contenute in qualsiasi applicazione Win-32. Il compilatore/ decom-

pilatore integrato in Resource Hacker consente di modificare direttamente l'aspetto ed il comportamento di applicazioni chiuse.

ResHack.zip

SLOC Metrics 2.0

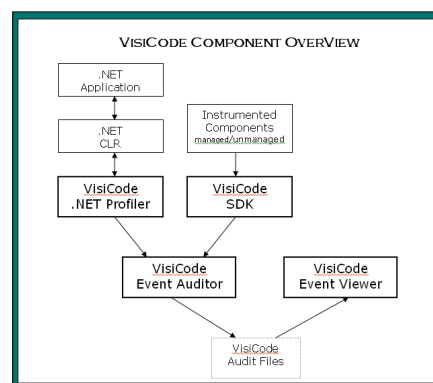
Misura la dimensione del codice sorgente

SLOC consente di conteggiare la dimensione del codice sorgente, secondo le raccomandazioni definire nel LOC (Physical Source Lines of Code). Il LOC è stato sviluppato dal Software Engineering Institute della Carnegie Mellon University e consente di avere una misura coerente e di paragonare più progetti. Nella versione licenziata, SLOC può scandire sorgenti di pressoché tutti i linguaggi: C++, C#, Java, HTML, Perl, Visual Basic ed altri ancora. Nel CD trovate una versione di valutazione che limita a trenta i file che è possibile inserire in un progetto. Il costo della licenza per la versione completa è di 90,00.

sloc_setup.exe

VisiCode 1.1

Analizza il comportamento e le performance di applicazioni .NET

Gli sviluppatori potranno utilizzare questa applicazione per testare in modo rapido ed intuitivo il codice che producono su piattaforma .NET. Attraverso una efficace rappresentazione grafica, VisiVode consente analizzare l'esecuzione di qualsiasi componente .NET in tempo reale, tenendo traccia delle relazione fra le classi e delle chiamate a più basso livello che maggiormente incidono sulle prestazioni.

visicode.zip

CodeJack 3.0

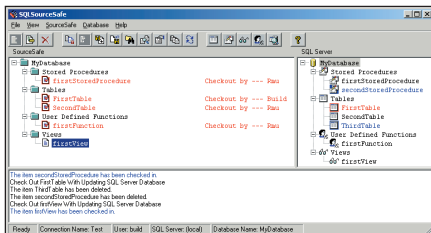
Gestire e condividere il codice che sviluppiamo

Un client per CodeShare che consente di gestire il processo di sviluppo ed il codice delle applicazioni all'interno di gruppi di lavoro. La nuova interfaccia, completamente ridisegnata, consente una più agile operatività.
Inoltre, grazie alla possibilità di essere integrato in Visual Studio 2003, gli sviluppatori potranno beneficiare dei vantaggi di CodeJack senza dover abbandonare l'ambiente cui sono abituati. Gratuito.

cj30.exe

SQLSourceSafe 2.2

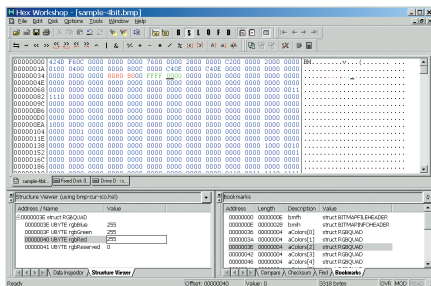
Gestisci il codice dei tuoi script SQL

In versione trial, un interessane prodotto per l'archiviazione e la gestione di script SQL. Si integra con SQL Server e Visual SourceSafe, offrendo agli sviluppatori un ambiente

coerente attraverso cui interagire con i database. La versione di prova valida trenta giorni, la distribuzione completa costa 279.00.

sqlsourcesafe_rel2.2_trial.exe

Hex Workshop 4.2

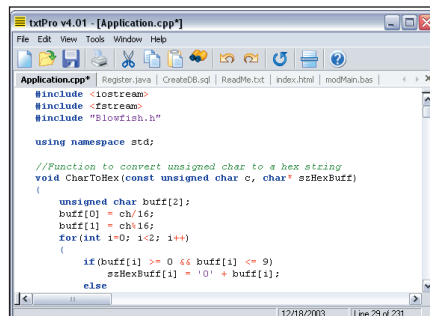
Un editor esadicimale semplice e ricco di funzionalità

Un tool per l'elaborazione di file in esadecimale che combina le capacità di un avanzato editor binario con la semplicità di un Word Processor. Hex Workshop consente di editare, tagliare, copiare, incollare, inserire e cancellare porzioni di file in esadecimale e

permette inoltre una facile pubblicazione del codice, sia in RTF che in HTML. Sono disponibili funzioni di goto, ricerca e sostituzione ed è possibile effettuare il confronto fra porzioni di file, calcolare il checksum ed altro ancora. Molto curata la sezione visuale dell'applicazione che consente di controllare e modificare i dati nel maniera più intuitiva. Grazie a questa rappresentazione la gestione di file esadecimali si presenta molto più semplice e veloce. Versione trial valida 35 giorni, il prezzo della versione commerciale è di 49,95.

hw32v420.exe

txtPro 4.01

Un ottimo editor testuale

La voglia di un editor tagliato su misura per noi non tramonta mai. Scrivendo centinaia di righe di codice al giorno, ogni programmatore ha bisogno di sentirsi perfettamente a suo agio con l'editor che utilizza.
txtPro può rappresentare una buona soluzioni per coloro i quali amino avere uno strumento ricco di funzioni e personalizzabile: è possibile scegliere il font con cui il testo verrà visualizzato; si può avere la visualizzazione del numero di riga accanto al codice; numerosi shortcut consentono di velocizzare l'accesso ai comandi di menu.

Da rimarcare la possibilità di effettuare riecerche testuali utilizzando le Regular Expression e la capacità di undo/redo infinito. Un'ultima nota per la presenza del "Repeat Command" una funzione che consente di reiterare l'ultimo comando impartito, velocizzando le operazioni più ripetitive. Versione di prova valida trenta giorni, il costo della versione completa è di 49,95.

txtProSetup.exe

Inno Setup 4.0.11

Installazioni professionali a costo zero

Assolutamente gratuito, Inno Setup ci aiuta a creare dei perfetti pacchetti di installazione per il nostro software. Tra le caratteristiche: interfaccia in stile Windows 2000, possibilità di compattare tutta l'installazione in un singolo exe, uninstaller, compressione, pieno supporto per l'installazione di risorse condivise come gli OCX e altro ancora. Collegandosi al sito dell'autore, è possibile scaricare il completo codice sorgente in Delphi.

isetup-4.1.1.exe

3IMPACT 2.0

Fai il tuo gioco

Un motore per la grafica tridimensionale che consente di realizzare giochi attraverso la commune sintassi C++. Acquistata la licenza (99.00) è possibile

distribuire liberamente i propri giochi, senza dover versare altre royalties. Tra le caratteristiche migliori si annoverano: la gestione della collisione fra oggetti, la sovrapposizione di elementi 3D, la simulazione di ruote con sospensioni, le ombre volumetriche, i sistemi di particelle (nebbia, polvere, fumo, ecc.), superfici riflettenti, suono tridimensionale ed altro ancora.

3impactdemo20.exe

BeWise Edelwise Basic Variables Freeware Version 1.0

I programmi VB comunicano con una semplice variabile!

Semplice e gratuito, BeWise consente di far interagire più programmi Visual Basic con una naturalezza incredibile. Attraverso una semplicissima interfaccia grafica si imposta il nome di una variabile. Da quel momento in poi, tenendo attivato BeWise, tutte le applicazioni VB in attività su quel PC riconosceranno quella variabile come propria variabile globale. Tutte le applicazioni potranno ispezionarne il contenuto e modificarla a piacimento. Con un po' di attenzione alle policy di accesso, sarà possibile costruire, con grande semplicità, sistemi di inte-razione anche complessi.

EdelwiseVBf.exe

LIBRERIE C++

WinPCap

Una Libreria Open Source per L'Analisi di Rete

Sviluppata all'interno di un progetto di ricerca del Dipartimento di Automatica e Informatica, Politecnico di Torino, questa libreria consente di emulare le

funzioni di libcap di Linux. Completa di codice sorgente, nel file compresso trovate anche WpdPack_3_01_a.zip che è il pacchetto di sviluppo di WinPCap (contiene .h e .lib per sviluppare in C utilizzando WinPCap)

\ WinPCap

CS-HTMLDiff Control 1.5

Analizza e visualizza le differenze fra documenti XML e HTML

Un controllo che consente di essere integrato sia in applicazioni server side che in applicazioni client e che permette di tenere allineate le differenze fra documenti diversi, siano essi XML o HTML. Completo di esempi.

csdiffctrl.zip

NCTVideoStudio ActiveX DLLs 1.3.3

Consente di convertire i filmati da uno standard all'altro.

Un pacchetto contenente ben 12 AcriveX che costituiscono un completo ambiente per il processo dei dati video. Sarà possibile convertire i file video da uno standard all'altro, creare video a partire da una sequenza di immagini, estrarre fotogrammi da una sequenza video, estrarre tracce audio dai filmati, mixare diverse tracce audio e farne la colonna sonora di un filmato, combinare o scomporre i video e molto altro ancora. Sono supportati i formati: AVI (DivX, XviD, MS MPEG4, e altri); MPEG (MPEG-1 ed MPEG-2 Video); WMV (Windows Media Video); RM (Real Networks Video, write only).

La versione distribuita è dimostrativa e presenta una schermata di avviso.

NCTVideoStudio.exe

GigaSoft ProEssentials 5.0

Un assemblaggio di componenti per il calcolo finanziario scientifico e ingegneristico

Un set di componenti, utili sia per applicazioni lato client che per oggetti lato server. Sono inclusi cinque distinti componenti: Graph, Scientific Graph, 3D Scientific Graph, Polar, and Pie Chart objects. Oltre mille funzioni diverse sono qui disponibili per la visualizzazione ottimale di testo e grafica. ProEssentials si presta ad essere utilizzato in campo scientifico così come in ambito finanziario.

pe5setup.exe

CSPrintingEngine 1.1

Ed ora si stampa!

Un componente COM che consente, con grande semplicità, di aggiungere alle nostre applicazioni la possibilità di stampare. Senza dover conoscere i segreti della stampa da Windows, potremo stampare qualsiasi cosa: immagini (sono supportai GIF, JPEG e BMP), tabelle, linee, rettangoli e testo di qualsiasi font e dimensione. Shareware.

CSPrintingEngine.zip

VintaSoftTwain ActiveX Control 2.1

Controllare le periferiche TWAIN

Un ActiveX che consente di pilotare scanner, Web Cam e qualsiasi altra periferica che supporti lo standard TWAIN. È possibile controllare pienamente il processo di acquisizione dell'immagine, decidere il formato in cui salvarla e, eventualmente, inviarla direttamente via FTP.

vstwain21.zip

SmartCode ViewerX VNC ActiveX Control 2.3.37

Il PC remoto è sotto controllo…

Un controllo ActiveX che mette a disposizione degli sviluppatori tutte le funzionalità del VNC attraverso un serie di metodi e proprietà assolutamente intuitivi.

viewerx.exe

LIBRERIE JAVA

Professional Applet Builder Collection 1.3 pop

Per creare annunci e menu

Un tool che consente anche ai meno esperti di creare effetti di grande livello, attraverso applet che implementano menu e sistemi di annunci. Supporta JavaScript e consente numerose personalizzazioni.

DropnNews.exe

jpcap

Cattura i pacchetti

Un set di classi java che fornisce una interfaccia per la cattura di pacchetti di rete.

Nel file compresso trovate anche una libreria ed un tool per la visualizzazione grafica del traffico di rete.

jpcap-0.4.zip

JpcapDumper

Visualizza i pacchetti rubati

Uno sniffer in Java che, poggiandosi si Jpcap, fornisce una rappresentazione grafica in tempo reale dei pacchetti catturati. Supporta i protocolli: Ethernet, IPv4/v6, TCP, UDP, ICMP, http.

JpcapDumper-0.2.jar

Hypermap

Le immagini in pugno

Un ottimo applet che consente di realizzare mappe "sensibili": sulla base di una qualsiasi immagine, è possibile indicare una serie organizzata di link che semplifica e rende più chiara la navigazione

hypermap.zip

Pets

Metti un acquario nel PC

Ideal per messaggi pubblicitari e banner, questo applet simula un piccolo acquario virtuale attorno alle lettere di qualsiasi testo si indichi. Le lettere che compongono il testo sono trasformate in elementi tridimensionali, dando la possibilità a dei piccoli pesciolini di effettuare delle simpatiche evoluzioni.

pets.zip

NCH News Ticker

La notizia è servita

Un simpatico ed efficace sistema per attirare l'attenzione sulla sezione news del tuo siti.

Con pochi semplici effetti, questo applet consente di realizzare un sistema di visualizzazione elegante e ben visibile.

42-Nch_news.zip

Birthday Announcer 1.0

Non solo compleanni

Un applet che effettua il matching fra messaggi e date, occupandosi di visualizzare il testo corretto nel giorno prestabilito.

ezbirthday.zip

Un viaggio nei meandri delle reti

Sniffer: spiare i segreti della Rete

Sulla scheda di rete del nostro PC, transitano pacchetti di dati che non sono indirizzati solo a noi. Scopriamo come recuperarli e ottenere preziose informazioni sui PC che ci stanno attorno.

Le applicazioni di questo articolo sono state create con la seguente configurazione PC:
- Pentium4 2.60GHz,
- 512Mb RAM

SISTEMA OPERATIVO:
- Windows XP Home SP1

SOFTWARE:
- Visual Basic 6 Enterprise Edition
- Libreria WinPcap del Politecnico di Torino (v3.01a)
- Controllo PacketX di BeeSync Technologies (v2.2)
- Libreria Java Jpcap di Keita Fujii (v0.4)
- Sun JDK (v1.4.2_03) con NetBeans (v3.5.1)

Le reti informatiche sono ormai entrate a fare parte della nostra quotidianità, a partire da internet che è alla portata di chiunque, fino alle reti LAN e WAN usate invece in contesti più professionali e specifici. La possibilità di mettere i computer in comunicazione tra loro ha aperto delle porte di cui non si immaginava neanche l'esistenza ed ha cambiato la vita di tutti. È chiaro che questa diffusione ed il successo delle tecnologie sottostanti alle reti di PC è dovuto in modo particolare ai tool software (browser internet, client di email, software per ftp, messenger vari, applicazioni di chat, etc.) che si sono resi via via disponibili e che consentono un accesso facilitato a sistemi che sono in realtà molto sofisticati e complessi: sebbene la maggior parte delle persone sia in grado di utilizzare tranquillamente Internet e buona parte delle sue funzionalità, quanti in realtà sanno cosa succede dietro le quinte? Non si può negare che una conoscenza tanto ampia sia di poca utilità per chi vuole solo godere dei frutti degli avanzamenti tecnologici nel campo dell'internetworking, ma per chi invece fa sul serio con l'informatica, la consapevolezza dei dettagli nascosti porta ad una maggior capacità di utilizzo degli strumenti che si hanno a disposizione. Non solo: sapere per filo e per segno cosa avviene all'interno delle nostre schede di rete ci permette anche di comprendere più a fondo i rischi di sicurezza a cui ci esponiamo con l'affacciarci sulla rete pubblica e le possibilità di difesa di cui possiamo disporre.
L'obiettivo di questo articolo è di costruire quelle basi che permettono di cominciare a camminare per il sentiero dei cavi *Ethernet* collegati ai nostri PC alla scoperta di che cosa effettivamente viaggia attraverso di essi quando navighiamo su Internet o compiamo in genere le nostre quotidiane passeggiate per la rete. Il punto di partenza per questo tipo di approccio è l'analisi del traffico di rete ad opera di strumenti detti *"sniffer"*, per cui il nostro lavoro sarà quello di scrivere appunto uno di questi tool. Al fine di capire cosa fa uno sniffer, dobbiamo ricordare che i dati che passano sulle schede di rete vengono inviati sotto forma di "pacchetti" e che il lavoro dello sniffer è proprio quello di visualizzare i vari pacchetti che una scheda di rete riceve prima che questi vengano ricomposti ad un livello più alto per essere processati dalle applicazioni comunemente usate per navigare e comunicare su internet.

IL CONTROLLO PACKETX

Per il nostro primo progetto ci avvarremmo di due librerie che potete trovare sul CD allegato: *WinPcap*

NOTA

COS'È UNO SNIFFER?

Lo sniffer è un'applicazione che si mette ad ascoltare, per così dire, su un dispositivo di rete: tutte le volte che tale dispositivo riceve od invia dati sul supporto fisico a cui è collegato (cavo Ethernet, linea telefonica, cavo USB), lo sniffer non fa altro che intercettare e memorizzare questa comunicazione, visualizzandola a video o loggandola in un file specificato. Tecnicamente parlando, uno sniffer non interferisce sul transito dei dati, ma si limita ad analizzarlo: quando i dati che transitano su un dispositivo vengono alterati nel loro percorso si parla più propriamente di spoofing se l'intento è quello di mascherare la propria provenienza o identità. Lo sniffing è un potente strumento di troubleshooting, di analisi dello stato di una rete, di verifiche di intrusioni, attacchi e violazioni. Ovviamente, nelle mani sbagliate, può diventare altresì un efficace quanto pericoloso strumento di recupero di nomi e password su reti locali a danno di certi protocolli (FTP, Telnet, POP3) in base ai quali queste informazioni preziose vengono passate totalmente in chiaro.

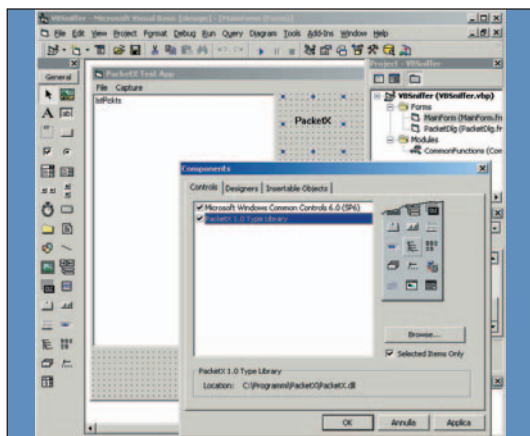



Fig. 1: *Il progetto VBSniffer e i componenti aggiuntivi da selezionare.*

(una serie di funzioni C che permettono la cattura dei pacchetti in transito sulla nostra scheda di rete), *PacketX* (un wrapper ActiveX di queste funzioni). Utilizzando il wrapper risulterà più semplice la stesura del codice dello sniffer in Visual Basic. Se volete ulteriori informazioni su queste librerie e dove reperirle, trovate tutto nel box degli *"Requisiti"*.
Il progetto da cui partiremo è una semplice applicazione .EXE, a cui però dobbiamo aggiungere i *Common Controls* di Microsoft per poter utilizzare la *ListView,* e ovviamente *PacketX:* i due componenti selezionati li vedete nella Fig. 1. Avremo due finestre: una principale che conterrà solo un menu ed una ListView dove raccoglieremo le informazioni che ci arrivano dalla scheda di rete (Fig. 2), ed una secondaria che ci mostrerà i dettagli di un singolo pacchetto che potremo selezionare dalla lista sulla finestra principale (Fig. 3). Il menu dell'applicazione ci consentirà di selezionare la scheda di rete su cui ascoltare e di iniziare o terminare la cattura dei pacchetti dalla scheda scelta. Il codice è abbastanza semplice, perché buona parte dei dettagli di basso livello sono nascosti allo sviluppatore grazie alla libreria *WinPcap* ed ulteriormente ad opera del

PacketX Test App

File Capture

No	Date	Size	Proto	Source	Dest
1	29/01/2004 16.28.55	60	Eth	0.0.0.0:0	0.0.0.0:0
2	29/01/2004 16.28.56	62	UDP	1.97.4.2:1985	224.0.0.2:1985
3	29/01/2004 16.28.57	42	Eth	0.0.0.0:0	0.0.0.0:0
4	29/01/2004 16.28.57	60	Eth	0.0.0.0:0	0.0.0.0:0
5	29/01/2004 16.28.57	62	TCP	1.97.4.122:4021	66.221.138.169:80
6	29/01/2004 16.28.57	60	Eth	0.0.0.0:0	0.0.0.0:0
7	29/01/2004 16.28.57	60	TCP	66.221.138.169:80	1.97.4.122:4021
8	29/01/2004 16.28.57	54	TCP	1.97.4.122:4021	66.221.138.169:80
9	29/01/2004 16.28.57	515	TCP	1.97.4.122:4021	66.221.138.169:80
10	29/01/2004 16.28.57	60	TCP	66.221.138.169:80	1.97.4.122:4021
11	29/01/2004 16.28.57	1514	TCP	66.221.138.169:80	1.97.4.122:4021
12	29/01/2004 16.28.57	1514	TCP	66.221.138.169:80	1.97.4.122:4021
13	29/01/2004 16.28.57	54	TCP	1.97.4.122:4021	66.221.138.169:80
14	29/01/2004 16.28.57	1514	TCP	66.221.138.169:80	1.97.4.122:4021
15	29/01/2004 16.28.57	54	TCP	1.97.4.122:4021	66.221.138.169:80
16	29/01/2004 16.28.57	1514	TCP	66.221.138.169:80	1.97.4.122:4021
17	29/01/2004 16.28.57	54	TCP	1.97.4.122:4021	66.221.138.169:80
18	29/01/2004 16.28.57	60	Eth	0.0.0.0:0	0.0.0.0:0
19	29/01/2004 16.28.57	602	TCP	66.221.138.169:80	1.97.4.122:4021
20	29/01/2004 16.28.57	54	TCP	1.97.4.122:4021	66.221.138.169:80
21	29/01/2004 16.28.57	1514	TCP	66.221.138.169:80	1.97.4.122:4021
22	29/01/2004 16.28.57	54	TCP	1.97.4.122:4021	66.221.138.169:80
23	29/01/2004 16.28.57	1514	TCP	66.221.138.169:80	1.97.4.122:4021
24	29/01/2004 16.28.57	54	TCP	1.97.4.122:4021	66.221.138.169:80
25	29/01/2004 16.28.57	62	UDP	1.97.4.3:1985	224.0.0.2:1985
26	29/01/2004 16.28.58	62	UDP	1.97.4.2:1985	224.0.0.2:1985
27	29/01/2004 16.28.59	60	Eth	0.0.0.0:0	0.0.0.0:0

Fig. 2: *La finestra principale dell'applicazione, con la ListView e il menu.*

wrapper *PacketX.* Brevemente, a livello di codice di interfaccia, il metodo di evento *Form_Load* viene utilizzato per resettare il contatore dei pacchetti ricevuti, impostare il flag di avvenuta scelta dell'interfaccia e creare le varie colonne della *ListView,* mentre nell'evento *Form_Resize,* ci assicuriamo che la lista occupi sempre tutto lo spazio disponibile della finestra e che le proporzioni tra le colonne siano mantenute.
Ancora, in *Form_Unload* interrompiamo un'eventuale cattura in corso (se il menu di *Stop* è abilitato significa che abbiamo fatto partire lo sniffing sulla scheda di rete) e scarichiamo dalla memoria la dialog box di informazione relativa al singolo pacchetto prima di terminare l'applicazione. Potete esaminare il codice di cui sopra, insieme ad un paio di altri metodi di semplice comprensione, direttamente dal sorgente del progetto disponibile sul CD. Noi andiamo invece ad occuparci ora della porzione più strettamente legata all'argomento in questione. Per fare ciò partiamo da una rapida analisi dell'API di *PacketX.* Quando siamo collegati ad una o più reti, il nostro computer deve possedere delle schede o adattatori che permettano fisicamente di far parte delle suddette reti. Per questo l'oggetto capostipite della libreria *PacketX* ci offre una collection di *Adapter* sotto forma di proprietà *Adapters,* cioè un elenco di tutte le interfacce di rete che sono presenti sul PC. Il metodo *reset* di PacketX inizializza tale elenco, ma viene invocato automaticamente all'istanziazione dell'oggetto, per cui è necessario invocarlo esplicitamente solo quando serve un aggiornamento perché – ad esempio – avete collegato qualche dispositivo di rete USB. Ogni oggetto della collezione è – dicevamo – un *Adapter,* da cui potete trarre varie informazioni sull'interfaccia di rete associata: il nome tecnico dell'interfaccia per il PC *(Device),* una descrizione più "umana' della stessa *(Description)* e dati run-time tipo il numero di pacchetti ricevuti *(PacketsRecv)* e persi *(PacketsLost),* la velocità di connessione in bps *(LinkSpeed),* gli indirizzi MAC *(HWAddress)* ed IP *(NetIP* e *NetMask),* ed altro ancora. Quello che interessa a noi nello specifico, però, è sapere quali pacchetti transitano sui vari adattatori e i dati che essi contengono. Per fornirci queste informazioni, il controllo PacketX utilizza un sistema ad eventi, ovverosia la nostra applicazione viene notificata di ogni pacchetto che un dato adattatore riceve.

```
Public Function ChooseInterface() As Adapter
Dim myPacketX As New PacketX
Dim v As Adapter
Dim msg As String
Dim itf As Integer
  msg = "Scegli un'interfaccia tra quelle disponibili"
                                      & vbCrLf & vbCrLf
  itf = 1
  myPacketX.Reset
  For Each v In myPacketX.Adapters
    msg = msg & itf & " : " & v.Description & vbCrLf
    itf = itf + 1
  Next
  itf = InputBox(msg)
  myPacketX.Adapter = myPacketX.Adapters(itf)
  MsgBox "Hai scelto : " & vbCrLf & vbCrLf &
myPacketX.Adapter.Description
  Set ChooseInterface = myPacketX.Adapter
End Function
```

LISTATO 1: Un esempio di come far scegliere un'interfaccia di rete all'utente.

Per lo sviluppo in Java è stato utilizzato *NetBeans* (prodotto gratuito e di ottima qualità, ex *Forte for Java*, ora fratello minore di *Sun Studio One*), ma è possibile lavorare con qualunque altro strumento di sviluppo.

L'installazione di *WinPcap* deve quindi precedere tutte le altre. *WinPcap* e *PacketX* possono essere installati utilizzando gli eseguibili di setup forniti, mentre per *Jpcap* è sufficiente la copia di un paio di file nella directory di libreria JDK o JRE come descritto nel file *README* dello zip di distribuzione.

NOTA

Le librerie *WinPcap* e *Jpcap* sono freeware e scaricabili rispettivamente da http://winpcap.polito.it e http://netresearch.ics.uci.edu/kfujii. *WinPcap* è una libreria che permette la cattura di pacchetti in transito sull'interfaccia di rete del proprio PC: contiene anche una DLL che offre delle funzioni compatibili con la popolarissima libreria libpcap di Linux, il cui obbiettivo è lo stesso.

Per prima cosa, dobbiamo avvisare l'istanza di PacketX su quale sia l'interfaccia che vogliamo tenere sotto controllo: per fare ciò basta settare la proprietà *Adapter* ad uno dei valori presenti nella collection *Adapters*. Il codice della funzione *ChooseInterface* utilizzata nella applicazione di esempio (vd *Listato 1*) si occupa di mostrare all'utente un elenco delle interfacce disponibili sul computer e permette una scelta, visualizzando tutti i dati di cui abbiamo parlato relativi all'interfaccia selezionata e ritornandone l'oggetto *Adapter* corrispondente come risultato della funzione. Sarà poi sufficiente impostare la proprietà *Adapter* di PacketX al valore ritornato in questo modo

```
<packetX_Instance>.Adapter = ChooseInterface
```

per selezionare un adattatore per la cattura, come avviene nel metodo *mnuFileAdapter_Click* del form principale del nostro sniffer. Come seconda cosa, una volta che abbiamo impostato in maniera simile a come indicato quale sarà l'adattatore su cui ascoltiamo, non ci resta che fare partire la cattura dei pacchetti di rete invocan do il metodo *Start* di *PacketX*. Dopo l'esecuzione di tale metodo, ogniqualvolta giungano dei dati sull'interfaccia che abbiamo richiesto verrà lanciato dal sistema l'evento *OnPacket* di PacketX, il quale a sua volta ci passa come parametro un'istanza di Packet, oggetto *ActiveX* che incapsula le intestazioni ed i dati che viaggiano nei vari pacchetti di rete.

```
` PACKETXLibCtl.IPktXPacket è l'interfaccia di base di PACKETXLibCtl.Packet
` utilizzata nella definizione dei vari elementi ActiveX di PacketX, per cui funzionalmente IPktXPacket e Packet sono equivalenti.
Private Sub pckCtrl_OnPacket(ByVal pPacket As PACKETXLibCtl.IPktXPacket)
  nCounter = nCounter + 1
  Dim clItem As ListItem
  Dim clSubItem As ListSubItem
  Set clItem = lstPckts.ListItems.Add(nCounter, , nCounter)
  Set clSubItem = clItem.ListSubItems.Add(, "Date", pPacket.Date)
  Set clSubItem = clItem.ListSubItems.Add(, "Size", pPacket.DataSize)
  Set clSubItem = clItem.ListSubItems.Add(, "Proto", ProtocolToString(pPacket.Protocol))
  Set clSubItem = clItem.ListSubItems.Add(, "Source", pPacket.SourceIpAddress & ":" & pPacket.SourcePort)
  Set clSubItem = clItem.ListSubItems.Add(, "Dest", pPacket.DestIpAddress & ":" & pPacket.DestPort)
  colPackets.Add pPacket
End Sub
```

LISTATO 2: Gestione della ricezione di un pacchetto

Tutto questo ci consente così di creare delle funzionalità che a fronte dell'arrivo di dati sulla rete analizzino e logghino tali informazioni per un uso successivo di diagnostica e risoluzione dei problemi legati alla stessa, ivi compresi quelli di sicurezza. Nel caso nostro, il metodo *OnPacket* del controllo PacketX, chiamato *pckCtrl*, è mostrato nel *Listato 2* e si occupa di incrementare il contatore dei pacchetti ricevuti *(nCounter)* e di aggiungere alla ListView *(lstPckts)* le informazioni essenziali sul nuovo pacchetto che è stato ricevuto (indirizzo e porta di origine e destinazione, data e ora, dimensione, vd. Fig. 2): queste variabili sono ovviamente estrapolate dalla sequenza di byte di cui consta il pacchetto stesso, sequenza che l'oggetto *Packet* esamina per noi alla ricerca di quei valori che poi ci fornirà sotto forma di proprietà *(SourceIpAddress, SourcePort, SourceMacAddress, DestIpAddress, DestPort, DestMacAddress, Date, OriginalSize)*, evitando così a noi il compito di dover decodificare le intestazioni di rete. Per avere una visione più completa dei dati giunti al nostro computer, sarà poi sufficiente fare doppio-click su una voce della *ListView lstPckts* per aprire la finestra del dettaglio (vd. Fig. 3) che visualizza praticamente tutte le proprietà offerte da *Packet*. Il codice del *Listato 3* è quello che si occupa della visualizzazione di tali proprietà e che mostra anche la sequenza di byte che compone la parte dati del pacchetto memorizzata come *Byte()* in *Data* (anche se è dichiarato come *Variant*) o *DataArray*. Per completezza, notate che il controllo *PacketX* può essere utilizzato in due forme differenti: come oggetto da instanziare via codice come nel *Listato 1*, oppure come controllo grafico *PacketXCtrl* come nel caso del form *MainForm*. In quest'ultimo caso, ovviamente, l'evento *OnPacket* sarà disponibile automaticamente come qualunque altro evento di controllo grafico, mentre se create l'oggetto via codice ricordate che per avere l'evento è necessario dichiarare la variabile relativa con la clausola *WithEvents* a livello di modulo, anche se nel nostro caso non è stato fatto perché non ci interessava catturare pacchetti nel metodo *ChooseInterface*.

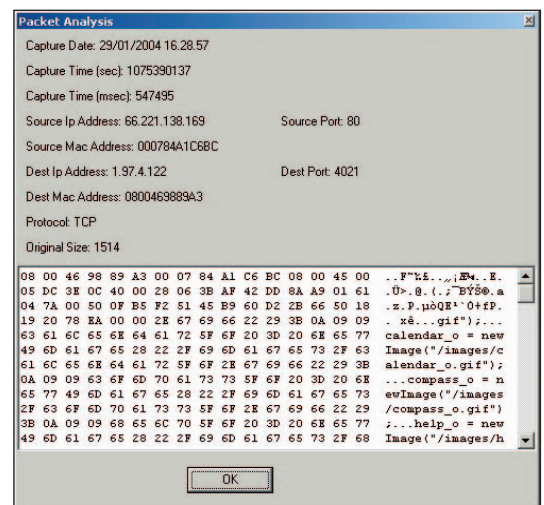



Fig. 3: **La finestra di dettaglio di un pacchetto catturato.**

WINPCAP IN JAVA CON JPCAP

Arrivati a questo punto avete a disposizione uno sniffer degno di tutto rispetto. Prima di discuterne però il funzionamento e di scoprire cosa possiamo imparare della rete, vediamo di rivedere i concetti legati alla sua creazione facendo uso questa volta però del wrapper *Jpcap* che mette sempre a nostra disposizione le funzionalità della libreria *WinPcap* (quella che avete utilizzato tramite *PacketX*), ma in

Java, permettendoci così di utilizzare questo linguaggio come base per catturare ed analizzare i pacchetti di rete. Fermo restando che la libreria *WinPcap* è scritta in C ed è tutto sommato quella forse meno adatta per chi vuole avvicinarsi ai protocolli di rete senza grosse conoscenze di quel linguaggio, il ponte creato da *Jpcap* rappresenta una soluzione interessante perché, se da un lato fornisce un accesso alla libreria originale sotto una piattaforma di sviluppo nota e diffusa, dall'altro si interfaccia ad essa in una maniera che non ne altera la sostanza e pare restare più a basso livello rispetto a *PacketX*. In altre parole, se è vero che con il controllo ActiveX di *BeeSync* si ha uno strumento di analisi del traffico di uso facilissimo, con *Jpcap* uniamo alla facilità d'uso una maggiore vicinanza, in termini di approccio, allo stile di *WinPcap*. Detto ciò, resta il fatto che sia PacketX che *Jpcap* sono delle astrazioni rispetto a *WinPcap*, e che in entrambi i casi ci viene risparmiata la necessità di lavorare con complesse strutture di dati, puntatori, handle di funzioni, e via dicendo: se avete intenzione di utilizzare in futuro la libreria in C (per esempio per poter scrivere applicazioni portabili anche sotto Linux), allora il mio consiglio è di prendere confidenza con *Jpcap* per un passaggio meno traumatico al mondo di *WinPcap* e *libpcap* (vd. ancora Box "*Requisiti*" su queste due librerie), tenendo a mente che comunque le funzionalità e capacità di PacketX e della controparte Java sono le stesse. Vediamo dunque come si scrive il nostro sniffer utilizzando questo nuovo API, ricordando intanto i punti salienti della cattura di pacchetti:

- **otteniamo una lista delle interfacce ed adattatori di rete della nostra macchina;**
- **selezioniamo una periferica di rete di cui ascoltare il traffico;**
- **iniziamo la cattura dei pacchetti;**
- **agiamo sulle informazioni che arrivano dal device;**

Il codice della classe Java che implementa lo sniffer è molto semplice ed è riportato per intero nel *Listato 4*. Per ottenere un elenco delle interfacce disponibili sul sistema abbiamo due metodi statici dell'oggetto *Jpcap* (package *jpcap*) che restituiscono un elenco dei nomi dei device (quelli utilizzati da Windows internamenti, di poco significato per noi) e un elenco della loro descrizione. Si tratta di *getDeviceList* e

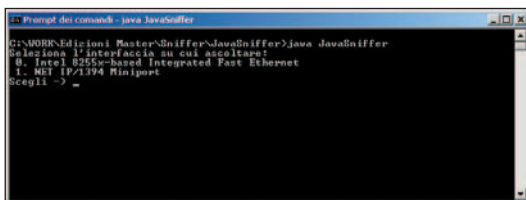



Fig. 4: ***La scelta dell'interfaccia nell'applicazione Java.***

getDeviceDescription rispettivamente, che restituiscono entrambi un array di stringhe. Il metodo *main* di JavaSniffer (la classe d'esempio) inizia proponendo all'utente un elenco degli adattatori di reti della macchina e richiedendo una scelta (vd. Fig. 4).
Una volta selezionata da parte dell'utente una descrizione di interfaccia, noi usiamo l'indice dell'array delle descrizioni per trovare il corrispondente nome di device e passarlo alla funzione statica *openDevice*, la quale imposta la scheda di rete indicata come base per l'ascolto dei pacchetti e restituisce un'istanza di Jpcap su cui lavorare. A differenza di quanto avviene in PacketX, qui dobbiamo impostare una serie di parametri relativi all'interfaccia che vogliamo monitorare: al di là del primo, che è il nome del device da attivare ottenuto da *getDeviceList*, gli altri sono, nell'ordine: il numero massimo di byte da catturare di ogni pacchetto, un flag booleano di settaggio della modalità promiscua (vd. Box 3), ed il timeout di inattività della scheda dopo cui interrompere la cattura (in millisecondi). Queste proprietà sono disponibili anche per l'oggetto *Adapter* di PacketX *(BPFSnapLen, HWFilter, ReadTimeout)*, oppure attraverso la finestra di proprietà custom di *PacketXCtrl* (vd. Fig. 5), ma a differenza di Jpcap non è necessario dargli un valore iniziale esplicitamente.

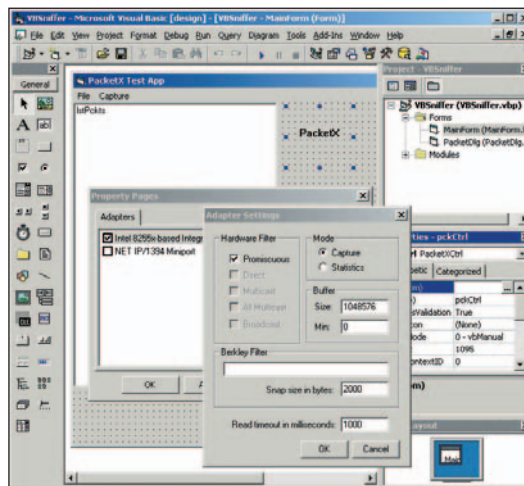



Fig. 5: ***Le proprietà avanzate dei device con PacketX.***

La dimensione massima dei dati da catturare (detta *SnapLen*) permette di migliorare le performance della cattura diminuendo la mole di dati che viene messa in gioco, ma ovviamente tralascia informazioni che potrebbero essere utili per un'analisi precisa e dettagliata delle trasmissioni sulla rete: siccome non esistono attualmente adattatori che lavorino con pacchetti più grandi di 65.535 byte, uno *SnapLen* di 65.536 vi garantisce di recuperare sempre i pacchetti nella loro interezza.
Se invece, per esempio, a voi servono solo le intestazioni, un valore decisamente inferiore per questa proprietà può migliorare le prestazioni della vostra

Jpcap **è un comodo insieme di classi Java che danno l'accesso alle funzioni di** ***WinPcap*****: quest'ultima deve quindi essere presente per un corretto funzionamento di** ***Jpcap*****. È consigliabile scaricare sempre l'ultima versione delle librerie, a meno di incompatibilità dichiarate con le versioni utilizzate per l'articolo. Il controllo ActiveX sviluppato da BeeSync è, invece, shareware ed è scaricabile presso** http://www.beesync.com **si tratta semplicemente di un wrapper delle funzionalità offerte da WinPcap, che deve quindi essere presente affinché il controllo funzioni.**

applicazione. Il timeout di cattura, invece, imposta un tempo limite di attesa in millisecondi: anche se non vi è attività sulla scheda aperta con *openDevice*, dopo il termine prestabilito il metodo di monitoraggio terminerà la sua esecuzione e tornerà al chiamante. Due valori hanno un significato speciale: lo zero indica che non si desidera un timeout di ascolto, mentre *-1* indica che il suddetto metodo dovrà tornare al chiamante non appena intercetti un primo pacchetto dall'interfaccia.

```
Public Sub SetPacket(oPacket As Packet)
Dim vByte As Variant
Dim sData As String
Dim nPosition, nColumns As Integer
  Label1.Caption = "Capture Date: " & oPacket.Date
  Label2.Caption = "Capture Time (sec): " & oPacket.TimeSec
  Label3.Caption = "Capture Time (msec): " & oPacket.TimeUSec
  Label4.Caption = "Source Ip Address: " & oPacket.SourceIpAddress
  Label5.Caption = "Source Port: " & oPacket.SourcePort
  Label6.Caption = "Source Mac Address: " & oPacket.SourceMacAddress
  Label7.Caption = "Dest Ip Address: " & oPacket.DestIpAddress
  Label8.Caption = "Dest Port: " & oPacket.DestPort
  Label9.Caption = "Dest Mac Address: " & oPacket.DestMacAddress
  Label10.Caption = "Protocol: " & ProtocolToString(oPacket.Protocol)
  Label11.Caption = "Original Size: " & oPacket.OriginalSize
  nColumns = 16
  For Each vByte In oPacket.Data
    If nPosition >= nColumns Then
      sData = sData + vbCrLf
      nPosition = 1
    Else
      nPosition = nPosition + 1
    End If
    If vByte <= &HF Then
      sData = sData + "0"
    End If
    sData = sData + Hex(vByte) + " "
  Next
  Text1.SelText = sData
End Sub
```

LISTATO 3: Visualizzare tutte le informazioni di un pacchetto.

Ma qual è il metodo che ci permette di iniziare il nostro procedimento di sniffing? In effetti abbiamo due possibilità: entrambe richiedono due parametri, che sono il numero di pacchetti da intercettare ed un'istanza di classe che implementi l'interfaccia *jpcap.JpcapHandler.* La differenza tra loro riguarda invece la gestione del valore di timeout, che è totalmente ignorato in *loopPacket,* mentre in *processPacket* è tenuto in considerazione in base alle specifiche che abbiamo visto poc'anzi. Per entrambi i metodi, infine, il numero di pacchetti da catturare prima di terminare la propria esecuzione può essere uguale a *-1* per non porre limiti da tale punto di vista. Nell'applicazione del *Listato 4* si è utilizzato *loopPacket* con un limite di pacchetti pari a *-1,* quindi per uscire dalla cattura sarà necessario *CTRL+C* o la forzatura della chiusura del processo. Per concludere, l'istanza di classe da passare a *loopPacket* o *processPacket* deve mettere a disposizione il metodo *handlePacket* dell'interfaccia *JpcapHandler.* Tale metodo verrà invocato per ogni pacchetto intercettato sul device di rete e riceverà come parametro un oggetto *jpcap.Packet,* che incapsula appunto i dati che sono in transito sul dispositivo di ascolto. È il parallelo dell'evento *OnPacket* di PacketX! Come potete vedere nel codice della classe, il compito di *handlePacket* è la traduzione in Java del lavoro di *pckCtrl_OnPacket* dell'applicazione in Visual Basic. Qui però ci avvaliamo del metodo *toString* di Packet, che è una funzione specializzata in ognuna delle sottoclassi, per cui a seconda del tipo di pacchetto avremo una riga di sintesi differente (Fig. 6).

NOTA

COSA SONO I PACCHETTI DI RETE?

Tutta la comunicazione di rete a cui siamo abituati è di fatto una grossa messa in scena a livello software: quello che avviene a livello più basso è che le informazioni che ci comunichiamo via Internet vengono smembrate in cosiddetti "pacchetti" che non superano praticamente mai i 1500 byte di dati per uno. Come avviene quindi che possiamo scaricare dei file di decine di Megabyte in un colpo solo? Molto semplicemente, questi Megabyte vengono spezzettati in tantissimi blocchetti da 1500 byte o giù di lì, inviati uno per uno verso il destinatario, dove saranno ricomposti – nell'ordine, ovviamente! – ricreando il file originale. Tutta questa macchinazione avviene in maniera del tutto trasparente ad opera dei vari protocolli che, a buccia di cipolla, costituiscono Internet: TCP o UDP al livello più alto, seguito da IP, e tipicamente Ethernet o PPP. Non ci credete? Allora costruite il vostro sniffer come mostrato in queste pagine, fate partire la cattura dei pacchetti sulla vostra interfaccia legata ad internet e richiamate una breve pagina web che conoscete: poi osservate la quantità esagerata di pacchetti che ricevete a fronte di quel poco codice HTML...

CONCLUSIONI PREGNANTI

Le reti informatiche sono una realtà molto complessa: semplificando di poco la dinamica di quelle a cui siamo più abituati, possiamo suddividerle in quattro livelli. Cominciamo dall'alto, dove troviamo

```
Output Window [Process Output]
Seleziona l'interfaccia su cui ascoltare:
 0. Intel 8255x-based Integrated Fast Ethernet
 1. NET IP/1394 Miniport
Scegli -> 0
Inizia la cattura dei pacchetti su Intel 8255x-based Integrated Fast Ethernet...

1075438473:489810
1075438473:489810 ARP REQUEST 08:00:46:98:89:a3(1.97.4.122) -> 00:00:00:00:00:00(1.97.4.1)

1075438473:490647
1075438473:490647 ARP REPLY 00:00:0c:07:ac:00(1.97.4.1) -> 08:00:46:98:89:a3(1.97.4.122)

1075438473:490664
1075438473:490664 1.97.4.122->207.46.107.113 protocol(6) priority(0)  hop(128)  offset(0) ident(16149) TCP 4169 > 1863 seq(2504443237) win(17228) ack 2017446778  P
50-4E-47-0D-0A                                                                                  : PNG..

1075438473:490765
1075438473:490765 ARP REPLY 00:00:0c:07:ac:00(1.97.4.1) -> 08:00:46:98:89:a3(1.97.4.122)

1075438473:660711
1075438473:660711 207.46.107.113->1.97.4.122 protocol(6) priority(0)  hop(48)  offset(0) ident(58331) TCP 1863 > 4169 seq(2017446778) win(17295) ack 2504443242

1075438473:661089
1075438473:661089 207.46.107.113->1.97.4.122 protocol(6) priority(0)  hop(48)  offset(0) ident(58332) TCP 1863 > 4169 seq(2017446778) win(17295) ack 2504443242  P
51-4E-47-20-34-38-0D-0A                                                                         : QNG 48..

1075438473:809165
1075438473:809165 1.97.4.122->207.46.107.113 protocol(6) priority(0)  hop(128)  offset(0) ident(16162) TCP 4169 > 1863 seq(2504443242) win(17220) ack 2017446786

1075438474:81695
1075438474:81695 10.48.2.157->224.0.0.1 protocol(2) priority(6)  hop(1)  offset(0) ident(61249)
11-64-EE-9B-00-00-00-00                                                                         : .dî.....

1075438474:810611
1075438474:810611 1.97.4.122->239.255.255.250 protocol(2) priority(0)  hop(1)  offset(0) ident(16163)
16-00-FA-04-EF-FF-FF-FA                                                                         : ..ú.ïÿÿú

1075438491:101500
1075438491:101500 1.97.4.122->1.253.128.10 protocol(17) priority(0)  hop(128)  offset(0) ident(16174) UDP 3010 > 53
00-59-01-00-00-01-00-00-00-00-00-00-03-73-64-6C-03-64-6C-73-03-63-6F-6D-00-00-01-00-01          : .Y..........sdl.dls.com.....

1075438492:95523
1075438492:95523 1.97.4.122->1.253.128.11 protocol(17) priority(0)  hop(128)  offset(0) ident(16175) UDP 3010 > 53
00-59-01-00-00-01-00-00-00-00-00-00-03-73-64-6C-03-64-6C-73-03-63-6F-6D-00-00-01-00-01          : .Y..........sdl.dls.com.....
```

Fig. 6: I pacchetti catturati con JavaSniffer.

Fig. 7: ***I vari livelli di rete compongono pacchetti a buccia di cipolla.***

ciò che conosciamo meglio: lo strato applicativo è in mano a protocolli quali *HTTP, FTP, SMTP, POP3, IRC*, e via dicendo… ormai ce ne sono una marea! Già a questo livello avviene una prima suddivisione in pacchetti più piccoli, ognuno con la sua intestazione che indica destinatario, mittente e sequenza dei vari pezzi. Ad un livello più basso (quello del trasporto) abbiamo invece i protocolli TCP e UDP, il primo orientato alla connessione stabile, il secondo ad efficienza e semplicità: i pacchetti del livello applicativo vengono incapsulati in pacchetti TCP o UDP, con un'ulteriore intestazione appiccicata in fronte. Segue poi il livello detto di network, dove regna quasi incontrastato IP (aiutato da ARP e pochi altri): il suo compito è quello di fornire un sistema di indirizzamento che colleghi i PC a livello mondiale: e anche qui, altra intestazione sopra al pacchetto di trasporto! Infine, al livello più basso, abbiamo i protocolli hardware di interazione tra la macchina e il cavo a mezzo di un dispositivo di rete: qui troviamo per esempio *Ethernet, Token Ring* (ormai poco diffusa se non in IBM), e PPP (le connessioni via modem). Le loro intestazioni collegano fisicamente le macchine tra loro, mentre ai livelli superiori abbiamo solo astrazione ed instradamento… La Fig. 7 mostra un pacchetto di rete in viaggio verso la sua destinazione: una volta giunto al termine, il software di sistema della macchina destinataria scorpererà i vari livelli fino a giungere a quello applicativo, che verrà così passato al codice dell'utente finale. Quando abbiamo in esecuzione uno sniffer, però, questo si infiltra in ascolto subito al livello più basso, quello *"physical"* ed è questa sua caratteristica che lo rende uno strumento di apprendimento insostituibile ed un potente mezzo di analisi di problemi e stato delle reti. Con un riferimento dei principali protocolli usati su Internet in una mano ed il mouse puntato su uno sniffer nell'altra, scoprirete un mondo molto affascinante e otterrete le conoscenze per sfruttare al meglio le potenzialità che le nuove tecnologie ci offrono. Sebbene abbiamo già sviluppato due sniffer completi, per ragioni di spazio non è stato possibile entrare nel merito di un'analisi dettagliata degli stessi a livello dei vari protocolli che si possono incontrare. Per chi però avesse fretta di capire cosa passa per la sua scheda di rete, ho ricercato ed accluso a questo articolo due prodotti sniffer che offrono già una segmentazione dei pacchetti per protocollo: si tratta di prodotti completamente gratuiti ed open-source, per cui potete anche divertirvi ad analizzare ed eventualmente migliorare le applicazioni.
Uno è basato direttamente su *WinPcap (analizer-.exe)* ed è un prodotto molto completo e complesso,

```
import jpcap.*;
class JavaSniffer implements JpcapHandler
{private static final int BYTES_PER_LINE=32;
public void handlePacket(Packet packet) {
  System.out.println();
  System.out.println(packet.sec + ":" +
                                  packet.usec);
  System.out.println(packet); // Questo
    forza una chiamata a packet.toString()
  StringBuffer sdata = new StringBuffer("");
  int i;
  for ( i = 0; i < packet.data.length; i++ )  {
    if (i !=0 && i % BYTES_PER_LINE != 0)
                          System.out.print("-");
    int b = (int)packet.data[i];
    if ( Character.isISOControl((char)b) )
      sdata.append('.');
    else
      sdata.append((char)b);
    b = (b >= 0 ? b : 256 + b);
    String hex = Integer.toHexString(b);
    if ( b < 16 ) hex = "0" + hex;
    System.out.print(hex.toUpperCase());
    if ( (i + 1) % BYTES_PER_LINE == 0 ) {
      System.out.print(" : " + sdata);
      System.out.println();
      sdata = new StringBuffer("");} }
  if ( i % BYTES_PER_LINE != 0 ) {
    sdata.insert(0," : ");
    for ( ; i % BYTES_PER_LINE != 0; i++
                    ) sdata.insert(0,"   ");
    System.out.print(sdata);
    System.out.println(); }}
public static void main(String[] args)
                  throws java.io.IOException {
  String[] lists = Jpcap.getDevice
                            Description();
  System.out.println("Seleziona l'interfaccia
                      su cui ascoltare:");
  for ( int i = 0; i < lists.length; i++ ) {
    System.out.println(" " + i + ". " +
                            lists[i]); }
  int n;
  do {
    System.out.print("Scegli -> ");
    n = System.in.read();
    n -= 48;
    System.in.skip(Long.MAX_VALUE);
  } while ( n < 0 || n >= lists.length );
  System.out.println("Inizia la cattura dei
pacchetti su " + lists[n] + "...");
  Jpcap jpcap =
Jpcap.openDevice(Jpcap.getDeviceList()[n
], 1000, false, 20);
  jpcap.loopPacket(-1, new
JavaSniffer());}}
```

LISTATO 4: Lo sniffer in Java con interfaccia testuale.

mentre il secondo richiede anche Jpcap perché è in Java *(JpcapDumper-0.2.jar)* ed è uno sniffer più simile ai nostri, ma con un dettaglio più elevato di studio dei dati dei singoli pacchetti. Spero che li troviate utili, e che possiate mettere a frutto quanto letto in queste pagine! Alla prossima…

Federico Mestrone

NOTA

LA MODALITÀ PROMISCUA DELLE INTERFACCE

Sulle reti locali basate su Ethernet avviene che per specifica del protocollo di comunicazione al livello più basso (quello proprio dei cavi e delle schede), tutte le macchine collegate ad uno stesso hub (il punto di smistamento della rete locale) ricevono tutti i pacchetti che vi passano attraverso, scartando quelli non destinati a sé stesse e trattenendo per successive elaborazioni quelli che erano invece intesi per sé. In sostanza, se l'indirizzo MAC di destinazione non coincide con il proprio il pacchetto viene ignorato, e questo avviene già al livello del driver della periferica. Esiste però la possibilità di porre il dispositivo hardware in quella che è detta "modalità promiscua", che prevede la lettura di qualunque pacchetto viaggi sul cavo, anche quelli indirizzati a qualcun altro. In questo modo, con applicazioni di tipo sniffer si può avere una visione di tutte le comunicazioni che avvengono sulle macchine che fanno fisicamente parte della nostra rete: questo ovviamente non sempre è positivo!

Pacchetti di installazione fatti in casa

Installer in Java: costruiamolo assieme

Un'applicazione è composta di molte parti: immagini, dll, file dati e tutte le numerose classi che vanno a definire progetti di qualsiasi complessità rendono necessario l'utilizzo di un installer…

Alzi la mano chi non ha mai sperimentato installazioni terminate con messaggi di errore incomprensibili o sequenze interminabili di spegnimenti e riavvii. Lo scenario si fa ancora più cupo se il povero sviluppatore si deve fare carico di garantire l'esecuzione su sistemi operativi con lingue diverse dall'italiano.
Che fare? Ebbene ci sono due possibili soluzioni: o affidarsi allo strumento standard Microsoft per la creazione di script di installazione, o scegliere uno strumento commerciale come *InstallShield*, *InstallAnywhere* etc. La soluzione Microsoft è sì gratuita, ma permette una installazione solo in italiano, inoltre l'installazione non è personalizzabile (ad esempio, è impossibile far eseguire procedure dall'installer), senza contare che l'installazione risulta spesso difficoltosa. Le soluzioni commerciali sono spesso potenti e personalizzabili, danno la possibilità di installare su più piattaforme, ma sono piuttosto costose e presentano difficoltà di upgrade tra le diverse versioni di Windows. Può succedere che la soluzione commerciale acquistata a caro prezzo per Windows 2000 funzioni male o non funzioni affatto in Windows XP.
In definitiva, chi scrive non ha saputo resistere alla tentazione di svilupparsi un installer "home made" che rispondesse ai seguenti requisiti:

1. Alternativa efficace all'installer offerto di corredo al Visual Basic, in grado di risolvere le lacune di quest'ultimo.
2. Installer facilmente configurabile, in grado di installare procedure sia in Java che Visual Basic in ambiente Windows.
3. Registrare l'installazione nel Windows Registry
4. Creare le opportune icone sul desktop e nella cartella programmi.
5. Generalizzare l'installazione nelle lingue inglese, francese, tedesco.
6. Permettere l'aggiornamento via Web dell'eseguibile e/o della base dati.

QUALI STRUMENTI UTILIZZARE

Java è un punto di riferimento per le sue doti di chiarezza e ci viene in contro con una ricca serie di classi, tutte di pubblico dominio, per arricchire l'installer. Inoltre, la procedura di installazione può essere avviata da una JVM presente sul CD di installazione.
L'elenco delle risorse richieste dal nostro applicativo da installare sono indicate nel file *SETUP.LST* generato dall'installer di corredo al Visual Basic. Questo file contiene infatti l'elenco e la relativa versione delle dll o componenti OCX richiesti dall'applicativo che vogliamo installare.
Le classi Java aggiuntive da usare nel nostro software di installazione sono:

Jconfig 2.2 : *Jconfig* è una libreria cross-platform che aggiunge nuove funzioni alle API standard Java. Permette eseguire applicazioni esterne, visualizzare pagine html,accedere alle informazioni tipo versione e data di creazione di ogni elemento nel file system. *Jconfig* è simile alle estensioni Microsoft di Java ma è multipiattaforma. Il software è free per lo sviluppo di applicazioni open source o shareware. Il link per il download è *http://www.simtel.net/product.download.mirrors.php?id=54577.*
JNIregistry 3.13. *JNIregistry* è una interfaccia Java per le API di accesso al registro di Windows, Permette di accedere modificare e esportare le risorse del registro di Windows. Il software è di pubblico dominio compresi i sorgenti. Il link per il download è *http://www.gjt.org/download/time/java/jnireg/registry-3.1.3.zip.*
Jshortcut 0.4. *JShortcut* è un package Java con la relativa libreria JNI che permette di creare e leggere con programmi shortcuts ed elementi di menu items *(ShellLinks)* in ambiente Windows. Il software è disponibile, sorgenti inclusi, con licenza LGPL all'indirizzo: *http://www.alumni.caltech.edu/~jimmc/jshortcut/download/index.html.*

Esistono diverse soluzioni per ottenere un eseguibile partendo da uno o più jar. Posso citare due soluzioni:

Excelsior Jet
http://www.excelsior-usa.com/jet.html
soluzione commerciale, dall'interfaccia molto scenografica, ma non sempre efficace;

Jsmooth
http://jsmooth.sourceforge.net/
dall'interfaccia spartana ma davvero molto efficiente.

INIZIAMO L'OPERA

Possiamo ora definire lo schema della nostra applicazione. Nella prima form possiamo selezionare la lingua di installazione (Fig. 1). Nella seconda form possiamo scegliere la cartella dove installare il software (Fig. 2), mentre nella terza form possiamo fare i controlli necessari per una corretta esecuzione del programma da installare (Fig. 3).
I controlli necessari sono:

- La versione del sistema operativo. Ad esempio potremo impedire l'installazione su alcuni sistemi operativi. Questo può essere fatto agevolmente usando la *System.getProperty("os.name"*") che ritorna il nome del sistema operativo.
- La disponibilità dello spazio disco sull'unità di installazione.

La quarta form permette di avviare l'installazione vera è propria (Cfr. Fig. 4). Il file SETUP.LST generato dal Visual Basic è un formato non particolarmente comodo per gestire le informazioni su versione, data di creazione e locazione delle risorse necessarie. Useremo un formato più compatto e piu'generale, di 6 campi

1. **Nome file da installare**
2. **Path di destinazione** (directory Windows o cartella del programma etc.)
3. **Versione**
4. **Flag di registrazione:** se vero il file corrente viene registrato
5. **Flag di esecuzione:** se vero il file corrente viene eseguito

Un esempio di questo formato può essere:

```
Vp3260.ocx,windir\System32,6.0.22.33,true,false
```

Questa riga ci dice il componente ActiveX *Vp3260 .ocx* deve essere installato nella cartella System32 di Windows, deve essere della versione indicata e che deve essere registrato. Il nome dell'eseguibile da installare lo si legge convenzionalmente dall'ultima riga del file con la lista di installazione. Nel nostro caso

```
Eseguibile_test.exe,path_inst,5.0.0.1,false,false
```

Utilizzeremo comunque l'elenco dei file e componenti con le loro versioni, cartelle di installazione, presenti nel file *SETUP.LST,* come punto di partenza per la creazione della nostra lista di installazione.
Nel caso di installazione di una procedura Java, la lista di installazione è l'elenco dei file jar utilizzati nel progetto, assieme ai file di dati o file ausiliari eventualmente richiesti. La versione dei file Jar non può essere gestita in maniera analoga ai componenti Windows, ma possiamo capire quanto una risorsa Java sia aggiornata dalla sua data di creazione. Quindi, per ognuno dei file presenti nella lista di installazione, occorre:

1. Verificare l'esistenza del file nella cartella indicata nella lista di installazione
2. Se il file esiste già sul sistema acquisirne versione e data di creazione
3. Se il file non esiste o è di una versione troppo vecchia, copiarlo sul sistema
4. Registrare il componente se richiesto.
5. Eseguire il file, se richiesto

Al termine della copia/registrazione dei file sul sistema, occorre concludere l'installazione con le seguenti operazioni:

1. Ricavare la locazione delle cartelle *Desktop* e *Programs* attraverso il registro di Windows. La cartella *Desktop* è evidentemente il desktop del sistema, mentre la cartella *Programs* è quella in cui vengono create le icone dei programmi installati.
2. Creare le icone nelle posizioni ricavate precedentemente.

Fig. 1: La scelta della lingua.

PIÙ IN DETTAGLIO

Molte delle operazioni indicate sono poco usuali in Java e perciò vale la pena descrivere come affrontarle con qualche dettaglio in più. Cominciamo dall'operazione più banale: la copia dei file. Pare incredibile, ma in Java non esiste alcun metodo per copiare i file. Per risolvere il problema, occorre copiare da un *InputStream* a un *OutputStream*, byte per byte fino alla fine del file. Di seguito trovate la porzione di codice che illustra l'operazione

Fig. 2: Selezione della directory di installazione.

```
int chunkSize = buffSize;
  byte[] ba = new byte[chunkSize];
    // Lettura fino a EOF
    while ( true ) {
        int bytesRead = readBlocking (source, ba, 0,
```

JCONFIG

Il vostro autore si è trovato qualche tempo a dover escogitare una soluzione che permettesse di visualizzare una pagina HTML da una applicazione desktop. Alcune soluzioni andavano bene su macchine con kernel NT e non su 98/95, altre erano insopportabilmente lente o incomplete. Jconfig è stato la soluzione: con una riga di codice si visualizza la pagina voluta con velocità, efficienza e su tutte le piattaforme. Ma Jconfig fa anche altro: dovete accedere al file system e sapere la versione del file o quando è stato fatto o le eventuali restrizioni o diritti di accesso? Di nuovo Jconfig è la soluzione. Anzi in questo caso è la soluzione. Dovete eseguire degli applicativi esterni e la JVM ve ne nega la possibilità? Jconfig ha molte risorse al riguardo. Jconfig può essere scaricato all'indirizzo www.jconfig.com La documentazione non è certamente il punto di forza di Jconfig. È disponibile un forum ma non molto frequentato.

```
                                        chunkSize);
        if ( bytesRead > 0 ) {
           target.write(ba,
           0 /* offset in ba */,
           bytesRead /* bytes da scrivere */);
        } else {
           break; // hit eof }
     } // end while
```

La cosa si generalizza facilmente al download via Internet, nel caso dell'aggiornamento di un file della nostra installazione. In questo caso le risorse di Java per il networking ci sono di grande aiuto.

```
   URLConnection urlc;
   urlc = source.openConnection();
   urlc.setAllowUserInteraction(false);
   urlc.setDoInput(true);
   urlc.setDoOutput(false);
   urlc.setUseCaches(false);
   urlc.connect();
   long length = urlc.getContentLength();
   InputStream is = urlc.getInputStream();
   FileOutputStream fos = new
                           FileOutputStream(target);
   boolean success;
   if ( length < 0 ) {success = copy(is, fos);
   } else {success = copy(is, fos, length); }
   if ( ! success ) {return false;}
   is.close(); fos.close();
```

Altro argomento un po' ostico è l'esecuzione di comandi esterni. Cosa, quest'ultima, indispensabile per effettuare la registrazione dei componenti, eseguendo in maniera sincrona il comando Windows *regsvr32/s*. L'uso del comando standard di Java *Runtime.exec* è reso impossibile dal fatto che su Windows 98 non funziona correttamente. La soluzione viene offerta da *Jconfig:* fonte inesauribile di risorse per il programmatore Java. Di seguito sono riportate le istruzione "a prova di pallottola" per gestire e controllare l'esecuzione di un processo esterno usando *Jconfig.*. Chi scrive ha avuto non pochi problemi a trovare un esempio funzionante del genere. *Jconfig* è realmente potente e efficace ma non ha nella documentazione il suo punto forte.

```
  Thread pausa = new Thread("pausa");
File curDir =new File (cur_dir);
FileRegistry.initialize(curDir,0);
File exeToRun = new File(nome);
AppFile app_file = FileRegistry.createAppFile(exeToRun);
  AppCommand app_cmd = app_file.getCommand(
           AppCommand.kAppCommandOpenDoc);
      app_cmd.clearArgs();
      app_cmd.addArg((String) arg); //
      if (app_cmd != null)
      { if (fl_sincro == true)
         { AppProcess appProcess =
             app_file.performCommand(app_cmd,0);
      pausa.sleep(10000);
      while (appProcess.isRunning() == true) { }
    } else
    { // esecuzione processo
     AppProcess appProcess =
          app_file.performCommand(app_cmd,0); } }
```

Jconfig ci torna utile anche per sapere quanto spazio abbiamo a disposizione sull'unità selezionata e qual è la versione e la data di creazione di un file. Nel primo caso, otteniamo l'informazione con il metodo *getFreeSpace* della classe *DiskVolumes*. Il codice che estrae queste informazione è riportato di seguito

```
DiskVolume[] dv = null;
int num_fs =0;
      if (FileRegistry.isInited()==false)
      {  File curDir = new File(cur_dir);
         FileRegistry.initialize(curDir, 0);}
      dv = FileRegistry.getVolumes();
      if (dv == null)
         Trace.println("fs = null");
      for (int i=0; i<dv.length; i++)
      { System.out.println("Vol. n."+(i+1)+";
  Nome = "+dv[i].getDisplayName()+"; "+dv[i].getPrefix()
                +";"+"Capacita'="+dv[i].getFreeSpace());
        String str1 =path.substring(0,3).toUpperCase();
        String str2 =dv[i].getPrefix().toUpperCase();
        if (str1.equals(str2) == true)
        { disk_avail = dv[i].getFreeSpace();
          break; }
```

Per l'accesso alle informazioni sui file possiamo usare il metodo *getVersion* della classe *DiskObject*.
Il codice che acquisisce le informazioni sui file è:

```
   file_v = new File( nome );
   disk_o = (DiskFile)
             FileRegistry.createDiskObject( file_v, 0 );
   if ( disk_o == null )
      {  err_code="err_016";
         return; }
       vn = disk_o.getVersion();
       if ( vn == null )
         {  return;}
         System.out.println("File="+nome+"; Vers.
                        ="+vn.getVersionString()+";
         Maj="+vn.getMajorVersion()+";
                        Min="+vn.getMinorVersion());
```

IL REGISTRO

Se si vuole installare del software su Windows, l'uso del registro è indispensabile. Ciò significa che è indispensabile trovare una classe Java che si faccia cari-

co di interrogare il registro e modificarlo ove serva. Questa classe, come abbiamo detto, è *Jniregistry.* Dobbiamo interrogare il registro per ricavare le informazioni che ci servono, che sono

- **Locazione della cartella "Programmi":** informazione da leggere nella chiave *HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\Current Version\Explorer\Shell Folders\Programmi*
- **Locazione della cartella Desktop:** informazione da leggere nella chiave *HKEY_CURRENT_ USER \SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\Desktop*
- Eventuale installazione preesistente dell'applicazione che dobbiamo installare. Supponendo che il programma da installare si chiami *eseguibile_test.exw,* la chiave da cercare è *HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\eseguibile_test.exe.* Questa chiave deve essere creata al termine della installazione assegnandovi il path di installazione.

Vediamo due esempi di codice in cui si utilizza *Jniregistry* per l'accesso al registro: il primo ricava la posizione delle cartelle *Programmi* e *Desktop,* il secondo scrive il path del programma che stiamo installando.

```
String val_d="";
String val_p="";
RegistryKey reg_sub_key=null;
Registry reg = new Registry();
RegistryKey reg_top_key= reg.getTopLevelKey(
                                "HKEY_CURRENT_USER");
try
{ reg_sub_key= reg_top_key.openSubKey(
                        KEY_REG_SHELL_FOLDERS,
                        reg_top_key.ACCESS_READ);
  val_d=reg_sub_key.getStringValue("Desktop");
  val_p=reg_sub_key.getStringValue("Programs");}
catch (Exception e)
{ return ;}
try
{ reg_sub_key.closeKey();
  reg_top_key.closeKey();}
catch (Exception e)
{ return ; }
path_desktop = val_d;
path_programs = val_p;
```

Ed ecco il secondo esempio di scrittura della chiave con il path del nostro software in corso di installazione:

```
RegistryKey reg_sub_key=null;
RegStringValue val = null;
// Creazione chiave con path eseguibile
```

```
try
{
   RegistryKey newKey=
   Registry.HKEY_LOCAL_MACHINE.createSubKey(
   KEY_REG_SELFAN, "", RegistryKey.ACCESS_WRITE);
                              //Istanza oggetto chiave
   val = new RegStringValue(newKey,"");
   val.setData(path+"\\"+EXE_TO_INSTALL);
                                   //Scrittura valore
   newKey.setValue(val); //Assegnazione del
                                   valore alla chiave
   newKey.closeKey();
}
catch (Exception e)
{
   e.printStackTrace();
   return;
}
```

L'ultima cosa da fare, al termine dell'installazione è creare le icone con i collegamenti all'eseguibile installato, nelle cartelle *Desktop* e *Programs. Jshortcut* permette di fare facilmente tutto ciò con l'oggetto *JshellLink.* Vediamo subito il sorgente coinvolto:

```
// Icona eseguibile su desktop
try {
   JShellLink link = new JShellLink();
   link.setFolder(path_desktop);
   link.setName("Selfan");
   link.setPath(path+"\\eseguibile_test.exe");
   link.setWorkingDirectory(path);
   link.setIconLocation(path+"
                        \\eseguibile_test.ico");
   link.save();
}
catch (Exception e)
{
   e.printStackTrace();
   return;
}
// Icona eseguibile in cartella programmi
try
{
   JShellLink link = new JShellLink();
   link.setFolder(path_programs);
   link.setName("eseguibile_test ");
   link.setPath(path+"\\ eseguibile_test.exe");
   link.setWorkingDirectory(path);
   link.setIconLocation(path+"
                        \\ eseguibile_test.ico");
   link.save();
}
catch (Exception e)
{  err_code="err_034";
   return;}
```

Neri Alemanni

VBRUN

L'installer presentato in questo articolo sfiora soltanto il problema più ostico di tutte le attività coinvolte nel processo di installazione: l'aggiornamento delle dll necessarie alle applicazioni VB. La cosa e' ostica perché le dll coinvolte dovrebbero essere aggiornate riavviando il sistema in modalità dos. Diversamente, infatti, le dll non possono essere sostituite perché in uso dal sistema. Una possibile soluzione potrebbe essere forzare l'esecuzione di VBRun60.exe. Quest'ultimo programma, fornito da Microsoft, permette l'installazione e/o aggiornamento dell'ambiente run time del Visual Basic 6.0.

Controllo di applicazioni tramite VBA

AutoCAD 2004 programmarlo con VBA

Come programmare il più famoso e diffuso dei programmi CAD usando il linguaggio Visual Basic for Applications per estendere e integrare l'ambiente con applicazioni office.

AutoCAD offre da sempre un potente linguaggio di programmazione per scrivere funzionalità aggiuntive: l'*AutoLisp* (dialetto del *Lisp* che è un linguaggio di tipo "funzionale"). Purtroppo, la scarsa diffusione di tale linguaggio non permette di sfruttare le conoscenze acquisite in altri programmi, pertanto eventuali "sforzi" volti ad approfondirne l'uso sono destinati a trovare ben pochi altri sbocchi al di fuori di AutoCAD. Per fortuna, c'è anche la possibilità di scrivere macro usando il VBA (acronimo di *Visual Basic for Applications*). Oltre al fatto di essere un linguaggio di programmazione particolarmente semplice, il VBA ha il vantaggio di essere incluso in tutti i maggiori pacchetti presenti per la piattaforma Windows, in primis nelle applicazioni che compongono la suite Microsoft Office.

ACTIVEX AUTOMATION

In questo articolo daremo per scontata una conoscenza (anche minima) della sintassi del VBA, mentre approfondiremo l'uso di alcuni oggetti che sono messi a disposizione da AutoCAD. Qualsiasi applicazione che implementa un'interfaccia ActiveX Automation mette a disposizione un insieme di oggetti, ciascuno dei quali realizza una parte delle funzionalità dell'applicazione stessa. In pratica, è possibile realizzare funzioni e procedure (e anche macro, che non sono altro che procedure senza parametri) che fanno uso degli oggetti esposti, cosicché è possibile sfruttare tutte le funzionalità presenti nell'applicazione e crearne di nuove. Inoltre è possibile usare tali oggetti anche al di fuori del programma stesso, in qualsiasi altro ambiente in cui è possibile utilizzare il VBA (o il VB). In pratica, potremmo sfruttare le funzionalità di AutoCAD direttamente in Excel o Word (tanto per citare due altre applicazioni che supportano il VBA). Inoltre è disponibile un editor intergrato tramite il quale poter scrivere i programmi VBA.

CREARE UNA NUOVA MACRO

Per iniziare, mandate in esecuzione AutoCAD (faremo riferimento alla versione 2004, ma anche nelle versioni precedenti c'è il supporto per il VBA). A questo punto potete scegliere la voce di menu *"Strumenti > Macro"*: si presentano quattro diverse opzioni; per ora scegliamo la prima *("Macro")*. Appare una finestra di dialogo per gestire le macro presenti (inizialmente non ci sarà nessuna macro). Sul campo *"Nome della macro"* digitate il nome della nuova macro, per esempio "test", e poi premete il pulsante "Crea". Ora vi viene chiesto dove tale macro deve essere memorizzata (mostrandovi i disegni e i progetti aperti). Scegliete il disegno corrente. Verrà aperto l'editor integrato VB come mostrato in Fig. 2. In figura potete distinguere le varie parti dell'editor: sulla sinistra in alto viene mostrata la struttura del progetto (notate come la macro viene creata in un nuovo modulo, che di default si chiama "Module1"), sempre sulla sinistra ma in basso ci sono le proprietà degli oggetti selezionati nella finestra precedente (struttura del progetto) mentre sulla destra (in alto) c'è la finestra dove inserire il codice VBA. Attualmente in essa c'è lo scheletro della macro da creare. Infine, in basso a destra, c'è una finestra con le "espressioni di controllo", utili in fase di debug del codice. Per ora potete chiudere l'intera finestra dell'editor VB: ritornerete così alla finestra di AutoCAD. Per riaprire l'edi-

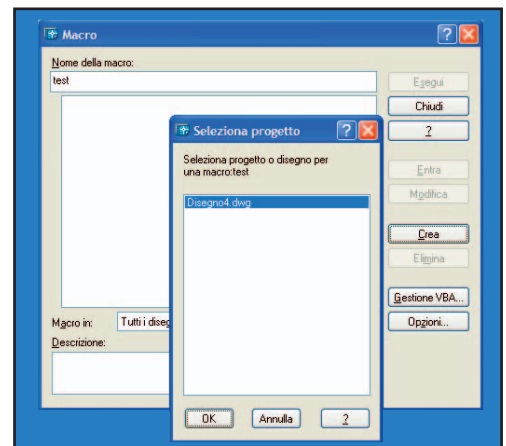



Fig. 1: Opzioni per la gestione delle macro VBA.

tor basta premere *Alt+F11* oppure selezionare *"Strumenti > Macro > Editor* di Visual Basic".

OGGETTI, PROPRIETÀ E METODI

All'interno della macro va scritto il codice VB voluto. Tale codice può far riferimento ad oggetti predefiniti in VBA oppure ad oggetti definiti da AutoCAD. L'oggetto di più "alto livello" è l'oggetto *Application* che rappresenta l'istanza di AutoCAD in esecuzione. Possiamo mostrare alcune sue proprietà, quali *Name, FullName* e *Version;* il risultato è visibile in Fig. 3 (ovviamente l'output è dipendente dalla versione di AutoCAD e dalla sua installazione). Ma come mandare in esecuzione la macro? In fase di test, la via più semplice è quella di restare nell'editor VB e premere il pulsante *"freccia a destra"* (è il simbolo *"Play"* usato normalmente nei lettori CD o VHS!); se invece siete nella finestra di AutoCAD potete scegliere *"Strumenti > Macro > Macro"*, selezionare la macro da eseguire e premere il pulsante *"Esegui".* Si è visto come usare l'oggetto *"Application"* e alcune sue proprietà (che altro non sono che attributi o valori). Ci sono anche dei metodi, ovvero funzioni che fanno "qualcosa" sull'oggetto in cui sono invocate. Un esempio è la funzione *Quit* dell'oggetto *Application,* che tenta di chiudere l'istanza corrente di AutoCAD:

Fig. 2: **L'editor integrato del Visual Basic.**

```
Application.Quit
```

In pratica ha lo stesso effetto della selezione della voce di menu *"File > Esci".* Altri oggetti permettono di accedere a tutte le parti che compongono AutoCAD o i suoi disegni. Per esempio c'è l'insieme *Documents* che contiene tutti i disegni attualmente aperti. Con il seguente ciclo possiamo stampare il loro nome:

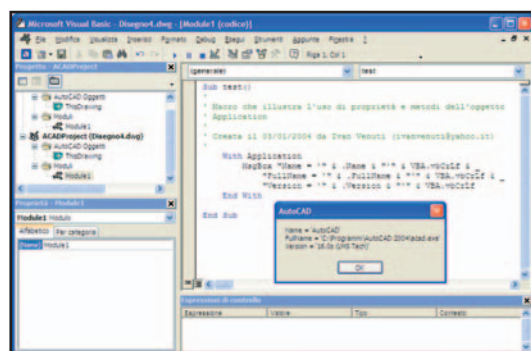



Fig. 3: **La macro creata e il suo output.**

```
For Each disegno In Application.Documents
   MsgBox disegno.Name
Next
```

DISEGNARE DA UNA MACRO

Attraverso l'oggetto *ActiveDocument* di *Application* (quest'ultimo, essendo l'oggetto di più alto livello, può essere omesso e faremo così d'ora in poi) possiamo riferirci al disegno attivo. Proviamo ora a disegnarvi delle figure per capire come possiamo disegnare da una macro VBA. A partire dall'oggetto *ActiveDocument* si ha a disposizione l'oggetto *ModelSpace* dal quale possiamo invocare diversi metodi per disegnare figure piane (per esempio *AddCircle* per disegnare un cerchio, *AddLine* per disegnare una linea) oppure figure 3D (*Add3DPoly* per disegnare un poligono tridimensionale). Supponiamo di voler creare una macro che disegni dei cerchi; la prima cosa da vedere è che parametri accetta *AddCircle.* Per farlo è possibile digitare, nell'editor, *".ModelSpace.AddCircle"* e poi premere *"spazio":* l'editor mostrerà i parametri voluti ed eventuali loro tipi. Siccome i nomi dei parametri sono auto-esplicativi, il più delle volte, è un aiuto sufficiente. Qualora non lo fosse, possiamo posizionarci con il cursore sopra la funzione e premere il bottone destro del mouse: dal menu a tendina va selezionato *"Definizione".* Ora appare la finestra "Visualizzatore oggetti" con informazioni sulla funzione. Se anche questo aiuto non è sufficiente si prema il pulsante destro (sempre sopra il nome della funzione dal visualizzatore oggetti) e si scelga "?". A questo punto appare un aiuto dettagliato sulla funzione, sui suoi parametri e sull'eventuale oggetto restituito (Fig. 4). Ora sappiamo che per disegnare dei cerchi è necessario specificare il centro del cerchio e il suo raggio. In tal modo, la funzione *AddCircle* crea e restituisce un oggetto *AcadCircle* che rappresenta il cerchio voluto. Su tale oggetto possiamo impostare ulteriori proprietà (come il suo colore) e, infine, disegnarlo usando la sua funzione *Update;* mentre il raggio è un numero reale (di tipo *Double*) il centro è un punto rappresentato da un array di numeri reali di tre elementi (che rappresentano le coordinate *X, Y* e *Z*). Ecco come disegnare un cerchio di coordinate (100, 100, 0) e raggio 30:

```
With ActiveDocument
   Dim centro(0 To 2) As Double
   raggio = 30
   centro(0) = 100
   centro(1) = 100
   centro(2) = 0
   Set cerchio = .ModelSpace.AddCircle(centro, raggio)
   cerchio.Update
End With
```

Sarebbe interessante permettere all'utente di specificare sia il centro che il raggio del cerchio. Per farlo possiamo utilizzare l'oggetto *Utility* del disegno *(ActiveDocument).* Esso possiede due metodi utili nel

nostro caso: *GetPoint* e *GetDistance*: il primo permette all'utente di selezionare un punto sul disegno e restituirlo come array delle sue coordinate; il secondo metodo permette all'utente di specificare una distanza a partire da un punto (che per noi può essere il punto selezionato come centro del cerchio!). Pertanto, una semplice macro che permetta di far selezionare centro e raggio e poi disegni un cerchio è:

```
Sub cerchio()
 With ActiveDocument
  Dim cerchio As AcadCircle
  centro = .Utility.GetPoint(, "Centro del cerchio")
  raggio = .Utility.GetDistance(centro, "Raggio")
  Set cerchio = .ModelSpace.AddCircle(centro, raggio)
  cerchio.Update
 End With
End Sub
```

Si noti che *GetPoint* e *GetDistance* hanno come secondo argomento un testo, il quale rappresenta il "prompt" per l'utente: tale messaggio verrà visualizzato nella finestra dei comandi sotto il disegno.
Prima di eseguire il metodo *Update* possiamo impostare il colore del cerchio, per esempio il blu, con l'istruzione:

```
cerchio.color = acBlue
```

L'oggetto di tipo *AcadCircle* possiede altre proprietà interessanti; per esempio ecco come mostrare l'area, la circonferenza e il raggio di un cerchio:

```
MsgBox "Area: " & tmp.Area & VBA.vbCrLf & _
        "Circonferenza: " & tmp.Circumference & _
           VBA.vbCrLf &  "Raggio: " & tmp.Radius
```

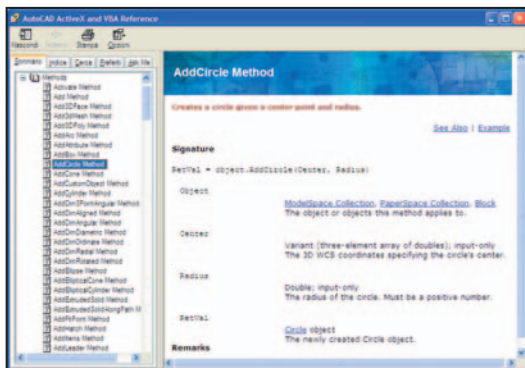



Fig. 4: ***Finestra di aiuto per la funzione AddCircle.***

ESEGUIRE IL RIEMPIMENTO

Un oggetto *AcadHatch* serve per impostare un riempimento di una o più figure. Esso viene creato dal metodo *AddHatch* sul *ModelSpace* di un disegno. Tale metodo ha i seguenti parametri:

- **PatternType:** valore di tipo *long* che specifica il tipo di pattern
- **PatternName:** valore stringa che definisce il nome del pattern da applicare
- **Associativity:** valore booleano (vale *True* o *False*) che indica se è associativo o meno
- **HatchObjectType** *(opzionale)***:** di default vale *AcHatchObject*, ma è possibile definirlo come *AcGradientObject* per specificare un gradiente.

Nel caso l'ultimo parametro sia *AcHatchObject* allora il *PatternType* (primo parametro) può assumere uno dei valori di *AcPatternType* (enumerazione), e il *PatternName* deve essere il nome del pattern. Per esempio, dopo aver impostato un cerchio e assegnatolo ad un array:

```
Dim tmp(0) As AcadCircle
Set tmp(0) = .ModelSpace.AddCircle(centro, raggio)
tmp(0).color = acRed
tmp(0).Update
```

ecco come impostare un riempimento di tipo solido di colore rosso:

```
Set nuovoHatch = .ModelSpace.AddHatch( _
        acHatchPatternTypePreDefined, "SOLID", True)
Set c1 = AcadApplication.GetInterfaceObject(
                              "AutoCAD.AcCmColor.16")
Call c1.SetRGB(255, 0, 0)
nuovoHatch.TrueColor = c1
```

Nel caso in cui *HatchObjectType* sia di tipo *AcGradientObject* allora il *PatternType* può assumere uno dei valori dell'enumerazione *AcGradientPatternType*, mentre *PatternName* può essere una stringa tra *LINEAR, CYLINDER, INVCYLINDER, SPHERICAL HEMISPHERICAL, CURVED, INVSPHERICAL, INVHEMISPHERICAL*, o *INVCURVED*. Per esempio ecco come si imposta un gradiente lineare tra i colori rosso e blu:

Fig. 5: ***L'output degli esempi di riempimento proposti nell'articolo.***

```
Set nuovoHatch = .ModelSpace.AddHatch(
  acPreDefinedGradient, "LINEAR", True, acGradientObject)
Set c1 = AcadApplication.GetInterfaceObject(
                              "AutoCAD.AcCmColor.16")
Set c2 = AcadApplication.GetInterfaceObject(
                              "AutoCAD.AcCmColor.16")
Call c1.SetRGB(255, 0, 0)
Call c2.SetRGB(0, 0, 255)
nuovoHatch.GradientColor1 = c1
```

```
nuovoHatch.GradientColor2 = c2
```

In entrambi i casi è importante che, dopo aver settato l'oggetto *hatch*, vengano eseguite le seguenti istruzioni:

```
nuovoHatch.AppendOuterLoop (tmp)
nuovoHatch.Evaluate
nuovoHatch.Update
```

I due esempi illustrati danno luogo all'output mostrato in Fig. 5. Sul CD-ROM, negli esempi, trovate una macro che permette di disegnare un numero qualsiasi di cerchi (specificando, con il mouse, centro e raggio) e ne esegue il riempimento con un gradiente. Il nome del gradiente e i colori di riempimento sono impostati in modo casuale; la macro (che si chiama *disegnaCerchi*) termina quando il raggio è 0 (ovvero il click per la distanza dal centro è sullo stesso punto del centro). In Fig. 6 un esempio della sua esecuzione.

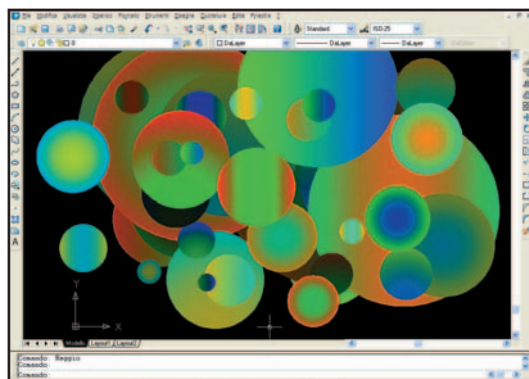



Fig. 6: Esempio di esecuzione della macro disegnaCerchi.

"EVENTI" ASSOCIATI ALL'APPLICAZIONE

È possibile scrivere delle macro la cui esecuzione avviene al verificarsi di determinati eventi sul documento, come la sua chiusura o apertura, il suo salvataggio o altri tipi di eventi. Dall'editor VB fate doppio click con il tasto sinistro su *"ThisDrawing"* nella fi-nestra del progetto (in alto a sinistra). Notate che poco sopra la finestra del codice ci sono due menu a tendina. I rispettivi valori di default sono *"(generale)"* e *"(dichiarazioni)"*.

Espandete la tendina a sinistra e selezionate *"AcadDocument"*. Automaticamente viene riempita la tendina di destra con i nomi degli eventi gestibili sul documento. Selezionando uno qualsiasi di tali eventi viene generata una procedura che gestisce l'evento selezionato. Selezionate, per esempio, il secondo evento *(BeginClose)* e poi modificate la procedura inserendovi l'istruzione: *MsgBox "Arrivederci!"* Ora quando chiuderete il disegno, apparirà il messaggio con il testo *"Arrivederci!"*.

Fig. 7: Finestra per la gestione VBA.

GESTIONE VBA

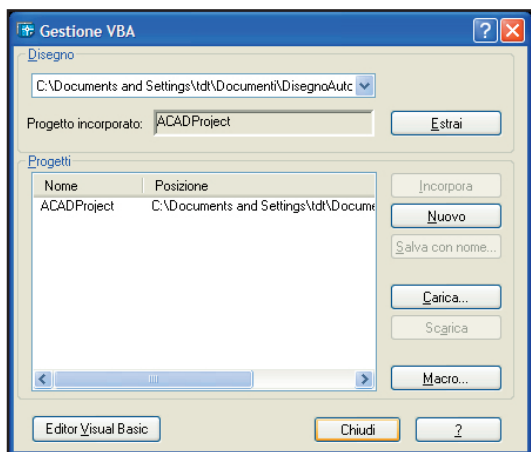
Ritornando alle opzioni del menu *"Strumenti > Macro"* prendiamo in considerazione *"Gestione VBA…"*. Scegliendo questa opzione si apre una finestra come quella mostrata in Fig. 7.
Vediamo il significato dei diversi campi e pulsanti:

Progetto incorporato: nome del progetto incorporato nel disegno attuale
pulsante "Estrai": permette di salvare come archivio a sé stante il progetto incorporato; una volta estratto, il progetto non risulta più incorporato
pulsante "Incorpora": a partire da un progetto presente nella lista a sinistra, lo rende incorporato
pulsante "Nuovo": crea un nuovo progetto
pulsante "Salva con nome…": permette di salvare (eventualmente rinominandolo) il progetto selezionato sulla lista di sinistra
pulsante "Carica": carica un progetto esterno nella lista
pulsante "Scarica": toglie dalla lista dei progetti quello selezionato
pulsanti "Macro" e "Editor Visual Basic": due delle opzioni attivabili da *"Strumenti > Macro.*

Sul CD allegato alla rivista trovate il file *"ProjectAutocadVBA.dvb"* che potete incorporare nei vostri disegni premendo *"Carica"* e selezionandolo. Esso contiene il codice completo delle macro descritte nell'articolo e, se decidete di utilizzarlo, vi consiglio di incorporarlo con il bottone *"Incorpora";* in questo modo farà parte del vostro disegno e non dovrete utilizzare sempre il CD.

CONCLUSIONI

Nell'articolo abbiamo visto come creare semplici figure, utilizzare oggetti di tipo hatch e come gestire i diversi progetti VBA. In un altro articolo vedremo come gestire i file *DXF* e come utilizzare AutoCAD da altre applicazioni attraverso l'automazione. Alla pagina *http://ivenuti.altervista.org/risorse/autocadvba.htm* potete trovare altre risorse e link per la programmazione di AutoCAD con il VBA.

Ivan Venuti

Web Services e interoperabilità nel nuovo Flash

Flash Mx 2004 incontra Google

Tra le molteplici novità di questa versione gli ingegneri di San Francisco hanno dedicato molta attenzione alla capacità dello strumento relative alla comunicazione con fonti dati esterni.

In questo articolo utilizzeremo il *WebService Connector* che fa parte del nuovo set di componenti contenuti nel pannello *Data Components* che comprende anche: *DataHolder, DataSet, RDBMResolver, WebServiceConnector, XMLConnector e XUpdateResolver.* Il componente *WebServiceConnector* gestisce i processi coinvolti nel passaggio dei dati tra l'applicazione Flash e l'applicazione server-side. Questa comunicazione avviene attraverso lo standard SOAP (*Simple Object Access Protocol*), in formato XML, cosa che si traduce in un'enorme semplificazione per lo sviluppatore che non deve più preoccuparsi di formattare le richieste del Web Service o interpretare i dati che gli vengono inviati, il componente gestisce il tutto in maniera trasparente ed automatica. L'applicazione che andremo a sviluppare consente di creare un motore di ricerca sul web appoggiandosi alle API di Google, sfruttando appunto il web service messo a disposizione dagli sviluppatori del più famoso motore di ricerca su Internet.

WEB SERVICES E SCREEN

I web services sono parti di software accessibili attraverso la rete per risolvere problemi specifici. La potenza dei web services risiede nel fatto che sono tutti sviluppati con tecnologie standard, indipendenti quindi dalla piattaforma: XML e Soap. Già dalla precedente versione, Flash aveva migliorato notevolmente le modalità di accesso e parsing dei dati esterni. Ora, attraverso l'introduzione del supporto per i web services, i passi fatti in avanti sono da gigante. Attraverso il nuovo pannello dei Web Services, al quale si accede attraverso il menu *Window* > *Development Panels* > *Web Services* (Fig. 2), è possibile definire un nuovo Web Service ed esplorare i parametri richiesti per il suo funzionamento ed esaminare i risultati che si ottengono dalla chiamata al servizio.

Fig. 1: *Il set di Data Components.*

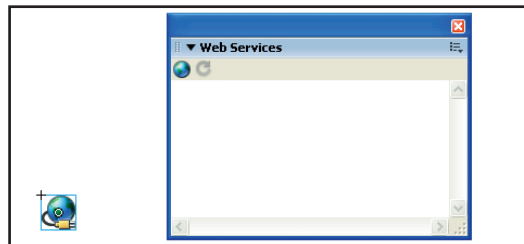



Fig. 2: *Il pannello Web Services: uno strumento di grande potenza.*

I PARAMETRI DEL CONNECTOR

Il *WebServiceConnector* offre allo sviluppatore un pannello di proprietà per poter interagire con le sue principali funzioni e proprietà, come mostrato in Fig. 3.

I parametri che possono essere modificati sono:

- **WSDLURL**: questo è il parametro che contiene l'url del file che definisce le operazioni del web service.
- **Operation**: dopo che il file è stato caricato que-

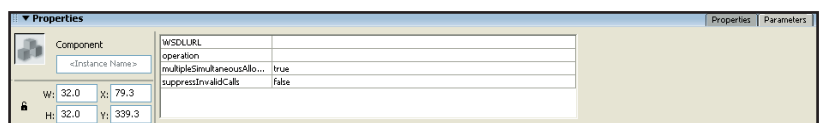



Fig. 3: *Il Property Panel del WebServiceConnector.*

WEB SERVICE

Applicazione identificata da un URI la cui specifica puo' essere definita, descritta e pubblicata, mediante meccanismi basati su XML e che supporta l'interazione diretta con altre applicazioni mediante messaggistica XML su protocolli Internet-based.

sto parametro permette di selezionare da un menu a tendina le operazioni che si possono eseguire sul web service.

- **multipleSimultaneousAllowed**: è un menù a tendina con i valori booleani *true* o *false* che determinano se il componente deve avanzare richieste al web service anche quando non sono state ricevute informazioni.
- **suppressInvalidCalls**: è un menu a tendina con i valori booleani *true* o *false* che restringe la possibilità di accettare data type diversi da quelli specificati dal file WSDL.
- **timeout**: è un parametro opzionale che permette di settare un tempo di risposta misurato in secondi passato il quale il componente cancellerà le richieste al servizio web se non avrà ottenuto nessun tipo di risposta.

NOTA

INTERFACCIA

L'interfaccia esprime una vista astratta di un ente computazionale, nascondendone l'organizzazione interna e quindi i dettagli di funzionamento. Quindi si focalizza sul funzionamento osservabile dell'entità. In altre parole, la struttura interna di un prodotto è inessenziale agli occhi del consumatore. Perché tutto funzioni è sufficiente assicurare il rispetto del contratto stabilito dall'interfaccia.

CLASSI DI OGGETTI E SCREEN

Oltre all'introduzione degli scrren e dei web services, la nuova versione di Flash, grazie alla nuova definizione del linguaggio ActionScript 2.0 permette di utilizzare il paradigma della programmazione ad oggetti (OOP) in maniera pressoché totale. Le keywords tipiche della OOP sono state implementate ed è finalmente possibile definire correttamente una classe di oggetti, applicare in maniera precisa i principi e i concetti dell'ereditarietà tra classi, estendere classi già esistenti, ecc. Relativamente a questi concetti, le *Form Applications* sono lo strumento dove risultano più evidenti tutti i vantaggi che si possono ottenere da questo nuovo approccio alla programmazione in Flash. Se create un nuovo file e scegliete tra le possibili opzioni una *Flash Form Application,* noterete facilmente che, dopo aver selezionato uno screen dell'applicazione, dal pannello properties, si può vedere il *Class Name* (di deafult *mx.screens.Form*), ovvero il nome della classe di oggetti ai quali viene associata ogni singola form. Già attraverso questa opzione è possibile sfruttare i vantaggi della programmazione OOP associando una classe che conterrà tutti i controlli relativi ad un preciso step dell'applicazione e mantenendo quindi separati il codice dalla *Graphic User Interface* (GUI) di un'appliczione.

STRUTTURA DELL'APPLICAZIONE

Innanzitutto definiamo lo scopo e l'architettura dell'applicazione che andremo a sviluppare: l'obiettivo finale è quello di creare un'interfaccia utente attraverso la quale sfruttare le API di Google e visualizzare i risultati della ricerca sul web. Per creare questa applicazione abbiamo bisogno di un account su Google in modo da sfruttare le API del famoso motore di ricerca (per ottenerle è sufficiente consultare l'indirizzo *www.google.com/apis/*) e di due classi all'interno delle quali definiremo i controlli dei tre step da cui sarà composta l'applicazione: *search, loading* e *result.* Oltre alla struttura interna che sarà formata da tre screen separati, inseriremo le classi associate ad ogni screen in una cartella *classes* e creeremo un'altra cartelle all'interno della quale inseriremo un'immagine che assoceremo alla nostra applicazione.

OPERAZIONI PRELIMINARI

Nella nuova versione di Flash, tra i documenti che si possono creare, ci sono anche gli *ActionScript File* che altro non sono se non dei file in formato ASCII con estensione *.as*. Creiamo due file distinti e salviamoli nella cartellina *classes* rispettivamente con i nomi *Search.as* e *Result.as.* La Fig. 4 mostra la *Start Page* di avvio del programma che ci permette di selezionare tra l'altro anche file Actionscript. All'interno di questi due files andremo ad estendere la classe *Form,* una delle classi *built-in* della nuova release del software, creando l'interfaccia utente e tutti i controlli ad essa associati direttamente via codice. Prima di procedere con la scrittura del codice, eseguiamo alcune semplici operazioni su un nuovo file del tipo *Form Application:* assegnamo come nome allo screen principale "google", creiamo tre screen a cui assegneremo rispettivamente i nomi *search, result* e *searching,* associamo ai primi due rispettivamente le classi *Search.as* e *Result.as* assegnando il percorso completo della classe dal pannello *properties* dopo aver selezionato lo screen e trasciniamo, posizionandoli al di fuori dello stage sullo screen principale, un'istanza del component *TextInput,* una del component *Button,* una del component *TextArea,* una del component *RadioButton* ed una del component *WebServiceConnector* assegnando un nome istanza solo a quest'ultima (ad esempio *google_ws*). Tra le operazioni preliminari che restano da esegui-

Fig. 4: *La Start Page iniziale.*

re, la più interessante è senza alcun dubbio quella che consiste nell'aggiungere un nuovo Web Service attraverso l'apposito pannello. Una volta aperto questo pannello è sufficiente cliccare sull'iconcina che rappresenta il globo posta in alto a sinistra affinché si apra una finestra di dialogo dove inserire l'indirizzo del servizio di cui vogliamo usufruire (in questo caso *http://api.google.com/GoogleSearch.wsdl*). Immediatamente dopo l'inserimento, Flash ci permette di esplorare (con l'ausilio del pannello) le operazioni disponibili per il Web Service specifico e i parametri da inviare e i valori restituiti per ogni singola operazione. L'operazione che faremo eseguire alla nostra interfaccia è *doGoogleSearch* e la imposteremo come predefinita direttamente dal pannello *Parameters* al quale accederemo dopo aver selezionato l'istanza, avendo scelto il WSDURL specifico per le API di Google. I parametri da inviare per effettuare correttamente una chiamata all'operazione *doGoogleSearch* sono:

- **key** - Utilizzato per inviare la chiave che si riceve da Google dopo aver richiesto l'accesso al suo Web Service.
- **q** - Utilizzato per inviare al servizio i vocaboli che devono essere utilizzati per effettuare la ricerca sul web attraverso Google
- **start** - Parametro attraverso il quale è possibile indicare l'indice del primo risultato che si vuole ricevere da Google
- **maxResults** - Valore attraverso cui si indica a Google il numero massimo di risultati che si vuole vengano restituiti per ogni richiesta
- **filter** - Valore Booleano da utilizzare per specificare alle API di Google se attivare o disattivare il filtro automatico del motore di ricerca sui risultati restituiti
- **restricts** - Parametro utilizzabile per limitare la ricerca effettuata da Google ad un determinato *Google Web Index*
- **safeSearch** - Valore Booleano attraverso il quale si può indicare al servizio se applicare o meno il filtro per i contenuti per soli adulti
- **lr** - Stringa attraverso la quale è possibile specificare il linguaggio dei documenti da ricercare (per tutte le sigle è possibile consultare il link *http://www.google.com/apis/reference.html#2_4*)
- **ie** - Parametro ormai considerato obsoleto attraverso cui specificare l'encoding dell'input inviato alle API
- **oe** - Parametro ormai considerato obsoleto attraverso cui specificare l'encoding dell'output inviato alle API

La finestra che ci permette di visualizzare i dati in ricezione ed invio del web service che stiamo consumando si trova nel pannello *Component inspector* trova nel tab *Schema*, come mostrato in Fig. 5:

LA CLASSE SEARCH

Una volta eseguite queste operazioni, aprite il file *Search.as* e iniziamo a definire la classe che gestirà il controllo del primo step della nostra applicazione.

```
import mx.controls.*;
class classes.Search extends mx.screens.Form{
```

Utilizzando la keyword *import*, rendiamo immediatamente disponibili all'interno della nostra classe tutte le classi che definiscono I nuovi component di Flash Mx 2004 (I file sono fisicamente presenti sulla nostra macchina in *C:\Programmi\Macromedia\Flash MX 2004\en\First Run\Classes\mx\controls*), mentre attraverso *class* dichiariamo la nostra classe (notate che prima del nome ho inserito il percorso con la sintassi a punto per rispettare il packaging delle classi) e la definiamo come estensione della classe *Form* dalla quale erediterà quindi tutte le proprietà e i metodi. Dichiariamo come *private*, e quindi disponibili solo per la classe all'interno delle quali vengono dichiarate e definite, tutte le proprietà della classe

```
private var input:TextInput;
private var performSearch:Button;
private var choice:RadioButtonGroup;
private var choiceAll:RadioButton;
private var choiceIta:RadioButton;
private var path:MovieClip;
```

Nella dichiarazione di ogni singola proprietà abbiamo applicato le nuove regole dello *strict datatyping* dichiarando il tipo di dato esplicitamente e forzando quindi la variabile/proprietà a contenere solamente il tipo di dato specificato. Specifichiamo successivamente il costruttore definendolo come pubblico e quindi accessibile anche dall'esterno della classe

```
public function Search(){
addEventListener("load", this);
addEventListener("reveal", this);
init.apply(this); }
```

Questa funzione viene richiamata ogni volta che si crea un'istanza della classe ed al suo interno non succede niente altro se non la registrazione dell'istanza stessa a due eventi specifici (caricamento e visibilità) e il richiamo di un metodo *init*, privato e quindi accessibile solo per la classe.
Il metodo *init* crea un clip filmato vuoto all'interno del quale definiremo gli elementi della nostra interfaccia e imposta come visibile lo screen associato alla classe

```
private function init():Void{
path = this.createEmptyMovieClip("__holder__", 0);
this.visible = true; }
```

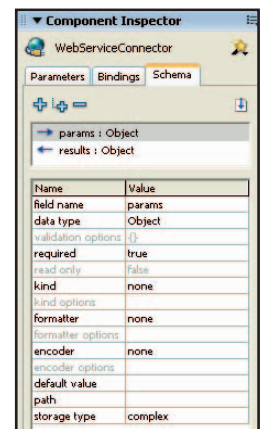



Fig. 5: *L'inspector consente di monitorare i parametri relativi all'invio e alla ricezione dei dati.*

La funzione *handleEvent* è il cuore della nostra classe. Quando lo screen viene caricato vengono creati e modificati gli elementi che compongono l'interfaccia, ovvero un campo di testo di input, due radio button per la selezione della lingua in cui effettuare la ricerca e un push button che invia la richiesta alle api di Google. Non appena viene rilevato il caricamento dello screen

```
if(obj.type == "load"){
```

viene creata nella proprietà *input* della classe un nuovo *TextInput* attraverso il metodo *createClassObject*, metodo della classe *UIObject* che restituisce un istanza della classe di oggetti specificata come primo argomento del metodo.

```
input = path.createClassObject(TextInput, "input", 2);
```

Visto che il metodo restituisce un riferimento all'istanza della classe *TextInput*, sarà possibile utilizzare i metodi *setSize* e *move* per posizionare correttamente il component sullo stage e tutte le proprietà della classe *TextInput* per limitare ad esempio il set carattere ammissibile nel campo di testo.

```
input.setSize(200, 25);
input.move(170, 185);
input.restrict = "a-z";
```

In maniera analoga si procede per il pulsante di invio e per i Radio Buttons che compongono l'interfaccia

```
performSearch = path.createClassObject(Button,
                                   "performSearch", 3);
performSearch.setSize(55, 25);
performSearch.move(input._x + (input.width -
                   performSearch.width), input._y + 28);
performSearch.label = "Search";
choiceAll = path.createClassObject(RadioButton,
                                        "choiceAll", 5);
choiceAll.setSize(100, 100);
choiceAll.move(input._x , input._y + 33);
choiceAll.label = "All the web"
choiceAll.data = "";
choiceAll.groupName = "choice";
choiceIta = path.createClassObject(RadioButton,
                                        "choiceIta", 7);
choiceIta.setSize(100, 100);
choiceIta.move(input._x + 80, input._y + 33);
choiceIta.label = "Italian";
choiceIta.data = "lang_it ";
choiceIta.groupName = "choice"; }
```

L'evento *reveal* scatta tutte le volte che lo screen viene reso visibile, ed è in questo momento che vengono associati gli eventi ad ogni elemento che compone l'interfaccia. In particolar modo, quando viene premuto il tasto di ricerca passeremo al *WebServiceConnector* i parametri necessari per effettuare la chiamata alle api di Google.

```
if(obj.type == "reveal"){
choiceIta.selected = true;
```

La nuova architettura dei component prevede l'utilizzo dei *Listener*, oggetti allocati nello stack di memoria in attesa di registrare un evento, ed è per questo che verrà creata un'istanza della classe *Object* e verrà registrata come listener dell'evento *click* associato al Button dell'interfaccia

```
var listToPb:Object = {};
listToPb.click = function(eventObj:Object):Void{
eventObj.target._parent._parent._parent.searching.visible
                                                = true;
eventObj.target._parent._parent.visible = false;
eventObj.target._parent._parent._parent.google_ws.params
                                                 = {};
eventObj.target._parent._parent._parent.google_
     ws.params.lr = eventObj.target._parent.choice.
                                    selectedRadio.data;
```

I parametri che passiamo al *WebServiceConnector* rispecchiano esattamente quelli necessari per il funzionamento delle api di Google.

```
eventObj.target._parent._parent._parent.google_
  ws.params.key = "2GRa0/tQFHJk+dRan0Lb6scMstJyKai3";
eventObj.target._parent._parent._parent.google_
                              ws.params.filter = true;
eventObj.target._parent._parent._parent.google_
                                  ws.params.start = 0;
eventObj.target._parent._parent._parent.google_
                            ws.params.maxResults = 10;
eventObj.target._parent._parent._parent.google_
                          ws.params.safeSearch = true;
eventObj.target._parent._parent._parent.google_
     ws.params.q = eventObj.target._parent.input.text;
```

Per far eseguire la chiamata sul Web Service è necessario utilizzare il metodo trigger della classe *WebServiceConnector;* quando si richiama il metodo vengono anche inviati tutti le variabili memorizzate nella *collection params* dell'istanza della classe.

```
eventObj.target._parent._parent._parent.google_
                                         ws.trigger();
```

La funzione che scatta quando il component riceve tutti i dati dal Web Service formatterà il testo e lo renderà visibile nel component che utilizzeremo nello screen successivo. La formattazione dei risultati avviene all'interno di un ciclo di *for…in* nel quale vengono gestiti anche i colori dei link in modo da ren-

API

Application Program Interface, interfaccia per programmi applicativi, risorse predefinite per la programmazione

dere del tutto simile a Google i risultati visualizzati all'interno della nostra interfaccia. Sempre utilizzando la stessa architettura basata sui listener, per intercettare e quindi visualizzare i dati prodotti dalle API di Google, memorizzeremo nella variabile *res* una funzione letterale che si occuperà della formattazione e della visualizzazione dei dati

```
var res = function (ev) {
for(var i in ev.target.results){
if(i == "resultElements"){
for(var k in ev.target.results[i]){
for(var j in  ev.target.results[i][k]){
var s = "";
s += "<a href=\"" + ev.target.results[i][k].URL +
      "\"><font size=\"+1\" color=\"#0000FF\"><u>" +
                ev.target.results[i][k].title + "</u></font>
                                             </a><br>";
s += ev.target.results[i][k].snippet + "<br>";
if (ev.target.results[i][k].summary != "")
{
   s += "<font color=\"#999999\">Description: </font>"
         +ev.target.results[i][k].summary + "<br>"; }
s += "<a href=\"" +ev.target.results[i][k].URL +
                 "\"><font color=\"#009900\"><u>" +
 ev.target.results[i][k].URL + "</u></font></a> - " +
                      ev.target.results[i][k].cachedSize;
s += "<br><br>";
eventObj.target._parent._parent._parent.result.__
                              holder__.test.text += s;}
    }
  }
}
eventObj.target._parent._parent._parent.result.visible
                                              = true;
eventObj.target._parent._parent._parent.searching.visible
                                              = false;
};
eventObj.target._parent._parent._parent.google_
                     ws.addEventListener("result", res);
}
performSearch.addEventListener("click", listToPb);
}
}
```

Passiamo ad esaminare la classe *Result* associata allo *screen result*. In maniera analoga alla precedente vengono importati e resi disponibili le classi contenute nel package *mx.controls* e dichiariamo come *private* le proprietà che utilizzeremo per memorizzare un riferimento ai componenti dell'interfaccia.

```
import mx.controls.*;
class classes.Result extends mx.screens.Form{
private var test:TextArea;
private var back:Button;
private var path:MovieClip;
public function Result(){
init.apply(this);}
private function init(){
this.visible = false;
path = this.createEmptyMovieClip("__holder__", 0);
addEventListener("load", this);
addEventListener("reveal", this);}
public function handleEvent(obj:Object):Void{
if(obj.type == "load"){
test = path.createClassObject(TextArea, "test", 2);
test.setSize(478, 265);
test.move(36, 70);
test.editable = false;
test.html = true;
test.wordWrap = true;
back = path.createClassObject(Button, "back", 5);
back.setSize(100, 22);
back.move(225, 370);
back.label = "Back";}
if(obj.type == "reveal"){
var listToBk = {};
listToBk.click = function(eventObj:Object){
eventObj.target._parent._parent.visible = false;
eventObj.target._parent._parent._parent.search.visible = true;
};back.addEventListener("click", listToBk);}
  }
}
```

Questo è il codice completo che permette alla nostra applicazione di girare. Tra le strade che ci hanno portato a sviluppare l'applicazione abbiamo volutamente scelto quella che richiedeva l'utilizzo di codice. È stata una scelta "onerosa" che però ci ha permesso di vedere da vicino ed usare le nuove funzionalità del programma e soprattutto il nuovo linguaggio Actionscript 2.
Riassumo brevemente le differenze con il vecchio linguaggio mettendo in evidenza le novità della seconda versione:

- fortemente orientato alla OOP (nuove *keywords : class, interface, packages..*)
- *"strongly typed"* (permette di definire in fase di dichiarazione il tipo di dato usato)
- *case sensitive*
- supporta i *class members private, public, static, ereditarietà* ed *interfaccie*

A chi intende avventurarsi sempre in maniera specifica nel programma e nel linguaggio del programma di casa Macromedia di cominciare ad utilizzare la nuova sintassi e soprattutto il nuovo approccio allo sviluppo, che si avvicina sempre di più ad un vero tool RAD come può essere Visual Basic.
Buono studio !

Marco Casario

NOTA

ISTANZA

Nella programmazione ad oggetti, l'istanza di una classe e' la realizzazione di un oggetto con le caratteristiche descritte della sua rappresentazione generale, la classe. Una classe e' un modello che definisce attributi (dati) e comportamenti (metodi) per le proprie istanze. Una classe puo' essere istanziata piu' volte per definire molteplici oggetti di quella classe. Nell'esempio viene riportata una classe in Java ed una sua istanza.

Come velocizzare lo sviluppo con Web Matrix

ASP.NET: l'accesso ai database

Uno strumento visuale come Web Matrix facilita, enormemente, lo sviluppo di pagine ASP.NET. In questo articolo vedremo come creare una pagina Web popolata con dati provenienti da un database.

Allo sviluppatore di pagine ASP.NET non risulterà probabilmente nuovo il nome Web Matrix: si tratta di uno strumento di sviluppo visuale per pagine ASP.NET, piuttosto completo anche se nella versione da noi provata (la 0.6) manca ancora il supporto per la creazione di Web Service (è presente di converso, nel menu *Tools*, un generatore di proxy per il consumo di Web Service). Il prodotto è scaricabile gratuitamente dal sito *http:// www.asp.-net/WebMatrix* ed il file di setup è un *.msi* (dunque facilmente rimovibile in caso di necessità) dal "peso" di soli 1,3 Mb, per cui è anche possibile portarlo presso il cliente su di un dischetto. In questo articolo vedremo come, tramite i wizard dello strumento e l'aggiunta di poche righe di codice, sia possibile ottenere una pagina ASP.NET con una tabella popolata dinamicamente con dati provenienti da un database. Il risultato che ci prefiggiamo di ottenere è quello di Fig. 1: una tabella opportunamente formattata, con una riga di intestazione e popolata con i dati provenienti, in questo caso, dal database *"Pubs"* fornito con Microsoft SQL Server 2000.

Cognome	Nome	Telefono	Indirizzo	Città	Stato	Zip Code
White	Johnson	408 496-7223	10932 Bigge Rd.	Menlo Park	CA	94025
Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618
Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705
O'Leary	Michael	408 286-2428	22 Cleveland Av. #14	San Jose	CA	95128
Straight	Dean	415 834-2919	5420 College Av.	Oakland	CA	94609
Smith	Meander	913 843-0462	10 Mississippi Dr.	Lawrence	KS	66044
Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705
Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301
Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	95428
Locksley	Charlene	415 585-4620	18 Broadway Av.	San Francisco	CA	94130
Greene	Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	37215
Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR	97330
Yokomoto	Akiko	415 935-4228	3 Silver Ct.	Walnut Creek	CA	94595
del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI	48105
DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN	46403
Stringer	Dirk	415 843-2991	5420 Telegraph Av.	Oakland	CA	94609
MacFeather	Stearns	415 354-7128	44 Upland Hts.	Oakland	CA	94612
Karsen	Livia	415 534-9219	5720 McAuley St.	Oakland	CA	94609
Panteley	Sylvia	301 946-8853	1956 Arlington Pl.	Rockville	MD	20853
Hunter	Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301
McBadden	Heather	707 448-4982	301 Putnam	Vacaville	CA	95688
Ringer	Anne	801 826-0752	67 Seventh Av.	Salt Lake City	UT	84152

Fig. 1: La pagina del nostro esempio Web popolata dinamicamente.

Fig. 2: Web Matrix funge anche da editor HTML visuale.

Apriamo dunque l'editor Web Matrix e, come scelta di progetto, scegliamo "*ASP .NET Page*" fra i template *"(General)"*; specifichiamo anche la cartella dove la pagina sarà creata, il nome della nostra pagina (ad es. *"DataAccess.aspx"*) ed il linguaggio che utilizzeremo, in questo caso VB.NET. Si apre la familiare schermata dell'editor, con le quattro viste (si notino i tab in fondo all'area di lavoro) *"Design"*, *"HTML"*, *"Code"* ed *"All"*. Sfruttiamo per prima cosa le capacità di Web Matrix come editor visuale per creare il template di tabella e l'intestazione. Lavoreremo, per ora, sulla vista *"Design"*. Dal menu *"HTML"*, clicchiamo sul comando *"Insert table…"* e scegliamo di inserire una tabella di una riga per sette colonne, in ognuna delle quali andremo a mettere l'intestazione; scegliamo anche una conveniente larghezza per la tabella (nel nostro esempio abbiamo adottato una larghezza di 800 pixel). Dalla finestra *"Properties"* in basso a destra nell'editor andiamo a settare le proprietà della tabella come lo spessore ed il colore dei bordi, l'allineamento del testo e così via (Fig. 2). Scriviamo anche le nostre intestazioni di colonna, formattandole opportunamente. Ora che lo scheletro della tabella è stato definito, è il momento di creare il codice per l'accesso al database. Vedremo

Fig. 3: **Utilizziamo il wizard “SELECT Data Method”, presente nella finestra “Code”, per svolgere la maggior parte del lavoro.**

come esistano due fasi distinte: un wizard per l’accesso ai dati vero e proprio, che creerà la maggior parte del lavoro costruendo una funzione con la stringa di connessione al database, l’accesso ai dati, la query per ottenere i risultati e restituendo un dataset popolato con i record, e poche righe di codice da scrivere “a mano”, in cui invochiamo la funzione costruita col wizard per avere un dataset le cui righe scorreremo per costruire dinamicamente il resto della tabella. Partiamo allora col nostro wizard. Portiamoci nella vista *“Code”* e nella toolbox di sinistra, sotto la voce *“Code Wizards”,* clicchiamo due volte sul comando *“SELECT Data Method”.* Nella finestra che si aprirà, lasciamo l’impostazione predefinita *“<New Database Connection>”* e nella lista *“Select a database type”* scegliamo *“SQL Server/MSDE Database”* (Fig. 3). La finestra successiva ci offre la possibilità di connetterci ad un’istanza di SQL Server: scegliamo il nome del nostro server dalla casella di riepilogo a discesa *“Server”,* scegliamo se usare l’autenticazione di Windows o quella di SQL Server fornendo in quest’ultimo caso nome utente e password, e selezioniamo il database al quale accederemo (*“Pubs”* nel nostro esempio).
La successiva finestra ci consente di costruire in maniera visuale la query al database (Fig. 4); selezioniamo la tabella e le colonne che vogliamo vengano memorizzate nel nostro dataset, e specifichiamo le eventuali condizioni di filtro nella finestra *“Where clause”.* Nella finestra successiva del wizard, possiamo testare il risultato della nostra query cliccando sul pulsante *“Test Query”;* verrà restituito, con un’interfaccia simile ad Access, l’insieme dei record che soddisfano la nostra query. Nell’ultima, importante finestra del wizard scegliamo un nome per la funzione che verrà generata (*QueryAuthors* è, nel nostro caso, un buon esempio) e scegliamo se la nostra funzione dovrà restituire un *DataSet* o un *DataReader* (Fig. 5). Nel nostro caso, dovendo semplicemente operare in lettura sequenziale su un insieme di record, sceglieremo di avere un *DataReader,* che è una struttura meno versatile, ma molto meno impegnativa in termini di risorse, del potentissimo *DataSet* di ADO .NET. Clicchiamo sul pulsante *“Finish”,* e troviamo definita la nostra funzione nella vista *“Code”.* Essa sarà la seguente:

Fig. 4: **Costruiamo, in maniera visuale, la query al nostro database.**

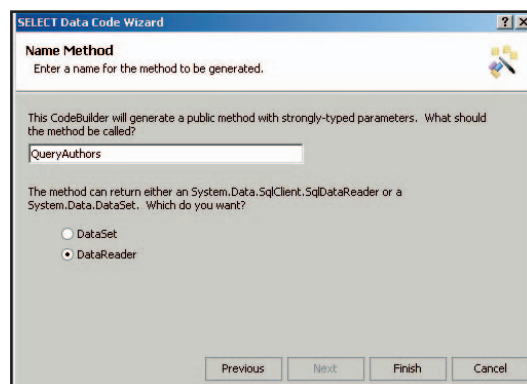



Fig. 5: **Diamo un nome alla funzione di accesso ai dati e scegliamo se questa restituirà un DataSet o un Data Reader.**

```
Function QueryAuthors() As System.Data.IDataReader
Dim connectionString As String = "server='(local)';
        trusted_connection=true; database='pubs'"
Dim dbConnection As System.Data.IDbConnection = New
        System.Data.SqlClient.SqlConnection(
        connectionString)
Dim queryString As String = "SELECT [authors].*
        FROM [authors]"
Dim dbCommand As System.Data.IDbCommand = New
        System.Data.SqlClient.SqlCommand
dbCommand.CommandText = queryString
dbCommand.Connection = dbConnection
dbConnection.Open
Dim dataReader As System.Data.IDataReader =
        dbCommand.ExecuteReader(
        System.Data.CommandBehavior.CloseConnection)
Return dataReader
End Function
```

Non dobbiamo preoccuparci dei complicati dettagli riguardanti la costruzione della stringa di connessione, la creazione della connessione vera e propria, la creazione della query e la sua esecuzione, ed il popolamento *DataReader* con i dati provenienti dal database.
È ora il momento di scrivere le poche righe di codice che sfruttano il *DataReader* per costruire la tabella. Ci portiamo nella vista *“All”;* la situazione in cui ci troviamo è quella di Fig. 6.

In primo luogo dobbiamo scrivere le direttive di import dei namespace che contengono le classi per l’accesso ad un database di SQL Server; stranamente, Web Matrix 0.6 non prevede alcun aiuto per la loro composizione.
Ad ogni modo, esse sono le seguenti:

```
<%@ import Namespace="System.Data" %>
<%@ import Namespace="System.Data.SQLClient" %>
```

e vanno subito sotto la direttiva di pagina, questa creata automaticamente da Web Matrix:

```
<%@ Page Language="VB" %>
```

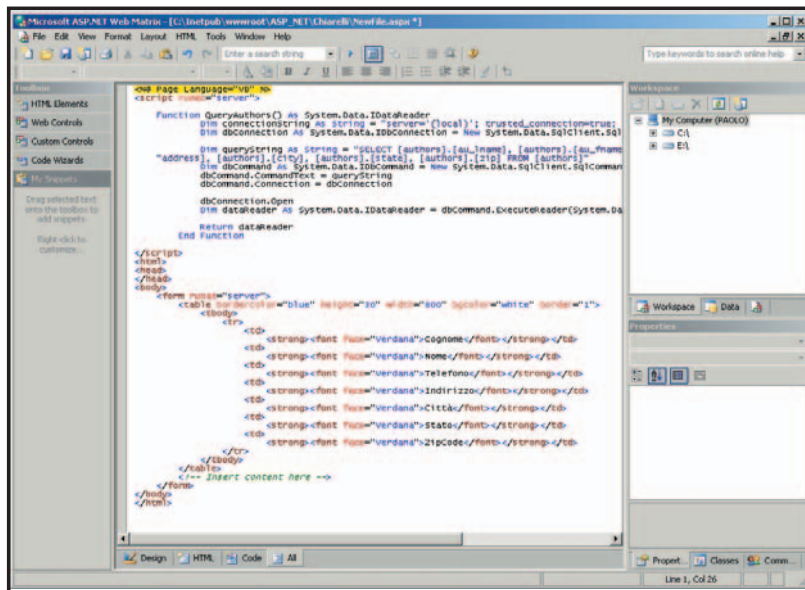



Fig. 6: **Fino a questo momento, non abbiamo dovuto scrivere neanche una linea di codice.**

Ci portiamo subito sotto il tag HTML *</tr>* che delimita la fine dell’intestazione della tabella, e digitiamo il codice evidenziato in Fig. 7:

```
    <%
        Dim dr As SQLDataReader
        dr = QueryAuthors()
        While dr.Read()
        Response.Write("<tr>")
        Response.Write("<td>")
        Response.Write(dr("au_lname"))
        Response.Write("</td>")
        Response.Write("<td>")
        Response.Write(dr("au_fname"))
                Response.Write("</td>")
                Response.Write("<td>")
                    Response.Write(dr("phone"))
                Response.Write("</td>")
                Response.Write("<td>")
                    Response.Write(dr("address"))
                Response.Write("</td>")
                Response.Write("<td>")
                    Response.Write(dr("city"))
                Response.Write("</td>")
                Response.Write("<td>")
                    Response.Write(dr("state"))
                Response.Write("</td>")
                Response.Write("<td>")
                    Response.Write(dr("zip"))
                Response.Write("</td>")
            Response.Write("</tr>")
            End While
        dr.Close()
        %>
```

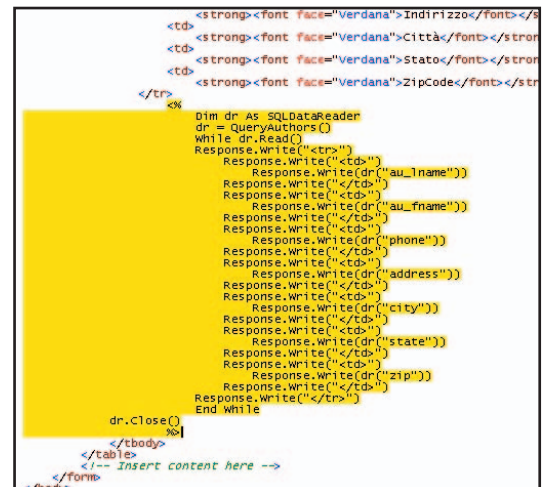



Fig. 7: **Quello evidenziato in giallo è tutto e solo il codice che dobbiamo scrivere manualmente.**

Creiamo un nuovo DataReader, *dr*, con una *Dim*, e vi assegniamo il DataReader creato dalla funzione *Query Authors*. Creiamo un ciclo While, la cui condizione di terminazione viene fornita dal metodo *Read()* del DataReader, che scorre automaticamente i record del DataReader ed assume valore *False* quando i record sono terminati. Peccato soltanto che Web Matrix manchi della funzione Intellisense di autocompletamento del codice….
Il codice rimanente è una semplice serie di *Response.Write()* che servono a creare le righe i cui dati sono disponibili accedendo per chiave al *Data Reader dr*. La nostra pagina di accesso ai dati è creata: in tutto, abbiamo dovuto scrivere solo 30 delle 82 righe di codice complessive, e se togliamo le ripetitive *Response. Write()* che creeremo a forza di copia-e-incolla, le righe effettive si riducono a 10. Non male per una versatile pagina Web che mostrerà dinamicamente dati provenienti da un database: volendo, possiamo ammirare il risultato del nostro lavoro anche premendo, sempre in Web Matrix, il tasto funzione *F5* e scegliendo se vogliamo utilizzare il Web Server incorporato in Web Matrix o se vogliamo creare una directory virtuale di IIS.

Paolo De Nictolis

Memorizzare i profili utente

Un Outlook in Java

(terza parte)

Tutti i programmi di posta elettronica includono meccanismi per la gestione dei profili e degli account. In questa parte del tutorial realizzeremo l'immagazzinamento su file system dei profili utente.

PREREQUISITI RICHIESTI

La realizzazione di quanto è mostrato in questo tutorial richiede, da parte del lettore, alcuni prerequisiti di base:

- **discreta conoscenza del linguaggio Java e della sua piattaforma;**
- **discreta conoscenza dei concetti di base del networking;**
- **discreta conoscenza dei meccanismi tipici di un servizio di posta elettronica.**

Con le prime due parti di questo tutorial, dedicato allo sviluppo di un client di posta realizzato con Java e basato sulle API JavaMail, abbiamo sviluppato alcuni importanti componenti software, che consentono l'invio e la ricezione della posta elettronica. Il codice già scritto ci tornerà senz'altro utile, anche se dovremo estenderlo e rivederlo prima di inglobarlo nella versione finale del nostro programma. Con questa terza parte del tutorial andremo a superare una delle limitazioni più stringenti tra quelle riscontrate sinora. Sia *MailSender* che *MailReceiver* (i nomi dati ai software realizzati nei mesi precedenti) pretendono che l'utente, ad ogni impiego, introduca i dati necessari per l'invio e la ricezione della posta. *MailSender,* infatti, vuole che ogni volta venga specificato l'indirizzo del server SMTP da impiegare per l'invio, mentre MailReceiver può controllare e ricevere la posta solo se l'utente specifica ad ogni utilizzo l'host POP3 cui collegarsi, insieme ai correlati dati di accesso indispensabili per autentificarsi e per accedere alla propria casella di posta elettronica. Un vero programma di posta elettronica, al contrario, deve ricordare i dati dei propri utenti, è evidente. In questa terza parte, dunque, ci prodigheremo nello sviluppo di un semplice e basilare meccanismo di gestione dei profili, scrivendo alcuni componenti che useremo, così come sono, nel programma finale. Si tratta di un argomento non direttamente in contatto con l'impiego delle API JavaMail, essendo un problema di ordinaria programmazione Java, ma che bisogna lo stesso affrontare e risolvere.

LA CLASSE PROFILE

Per prima cosa andiamo a progettare una classe destinata a rappresentare un profilo utente. Stabiliamo quali dati introdurre al suo interno. Per ora accontentiamoci delle informazioni basilari:

- **Nome visualizzato.** Quando l'utente inoltrerà una mail, il contenuto di questo campo sarà usato come nome visualizzato al destinatario. La *"Valeria Dal Monte"* di Outlook Express, per intenderci ;-) (questa la può capire solo chi si è ritrovato a configurare un sacco di volte gli account su questo programma per gli amici che non ne volevano sapere di usare altri software)
- **Indirizzo dell'host POP3** per la ricezione.
- **Username** per l'accesso al POP3.
- **Password** per l'accesso al POP3.
- **Indirizzo dell'host SMTP** per l'invio.

Questi cinque dati sono sufficienti per inquadrare la situazione più tipica di configurazione di un client di posta. Alla vostra volontà lascio la scelta di introdurre, ora o in seguito, altri campi. Ad esempio, alcuni host SMTP richiedono l'autenticazione dell'utente. Un'altra informazione che potrebbe essere salvata con il profilo è la firma da aggiungere automaticamente in coda alla posta in uscita. Trasformiamo in codice Java le ipotesi portate avanti sinora:

```
// Profile.java
// Lista degli import dalla libreria di Java.
import java.io.*;
// Questa classe permette il salvataggio ed il recupero
// dei profili su disco.
class Profile implements Serializable {
  private String pop3;
  private String smtp;
  private String username;
  private String password;
  private String name;
  // Il costruttore è privato, così non è possibile creare
  // dall'esterno degli oggetti profile.
  private Profile() {}
```

```
    // I seguenti metodi permettono l'accesso in sola lettura
    // alle proprietà del profilo.
    public String getPOP3() { return pop3; }
    public String getSMTP() { return smtp; }
    public String getUsername() { return username; }
    public String getPassword() { return password; }
    public String getName() { return name; }
    // Date le proprietà di un profilo, questo metodo crea
    // l'oggetto corrispondente e lo salva su disco.
    // Restituisce true se l'operazione di salvataggio riesce.
    public static boolean saveProfile(
        File f, String pop3, String smtp,
        String username, String password, String name) {
        Profile p = new Profile();
        p.pop3 = pop3;
        p.smtp = smtp;
        p.username = username;
        p.password = password;
        p.name = name;
        ObjectOutputStream out = null;
        try {
            out = new ObjectOutputStream(new
                                        FileOutputStream(f));
            out.writeObject(p);
            return true;
        } catch (Exception e) {
            return false;
        } finally {
            if (out != null) try {
                out.close();
            } catch (Exception e) {}
        }
    }
    // Dato un file, questo metodo lo legge e lo interpreta
    //come un profilo. Se l'operazione fallisce, restituisce null.
    public static Profile loadProfile(File f) {
        Profile p = null;
        ObjectInputStream in = null;
        try {
            in = new ObjectInputStream(new FileInputStream(f));
            return (Profile)in.readObject();
        } catch (Exception e) {
            return null;
        } finally {
            if (in != null) try {
                in.close();
            } catch (Exception e) {}
        }
    }
}
```

La classe *Profile* incorpora, sotto forma di campi privati, i cinque valori discussi sopra. L'accesso al loro contenuto è garantito in sola lettura, attraverso i metodi di tipo *get()*. È stato definito un costruttore privato privo di argomenti. Questa tecnica blocca la possibilità di creare istanze di *Profile* dall'esterno della classe stessa. La creazione di un profilo, pertanto, avverrà servendosi dei metodi statici *saveProfile()* e *loadProfile()*. Il primo crea un'istanza della classe *Profile* associandogli i dati forniti in argomento, quindi la salva su file usando la serializzazione dell'oggetto (per questo motivo *Profile* implementa *Serializable*). L'operazione restituisce *true* se tutto va a buon fine, oppure *false* se si riscontra qualche problema durante il salvataggio del file. Diametralmente opposto è il metodo *loadProfile()*, che come argomento accetta il riferimento ad un file e come risultato restituisce l'istanza di *Profile* caricata dal file system. In caso di problemi durante la lettura del file, *loadProfile()* restituisce *null*.

LA CLASSE PROFILEMANAGER

Ora che esiste un meccanismo per la gestione dei profili su file system, è necessario mettere a disposizione dell'utente uno strumento utile per la loro creazione, la loro modifica e la loro selezione. Per questo andremo ora a realizzare un componente che impiegheremo poi nel nostro client di posta, cioè la finestra di dialogo *ProfileManager:*

```
// ProfileManager.java
// Lista degli import dalla libreria di Java.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.io.*;
class
    ProfileManager
extends
    JDialog // E' una finestra di dialogo.
implements
    ActionListener, // Per gestire i pulsanti interni.
    ListSelectionListener, // Per gestire la lista interna.
    FileFilter // Per filtrare i file.
{  // Elementi Swing per la GUI.
    private JButton button1 = new JButton("Seleziona");
    private JButton button2 = new JButton("Modifica");
    private JButton button3 = new JButton("Elimina");
    private JButton button4 = new JButton("Crea nuovo");
    private JButton button5 = new JButton("Annulla");
    private JList profilesList = new JList();
    // Elenco dei file di profilo letti dal disco.
    private File[] profiles = null;
    // Riferimento al profilo selezionato dall'utente.
    private Profile selectedProfile = null;
    // Costruttore.
    public ProfileManager(JFrame owner) {
        super(owner, "Gestore dei profili", true);
    ...
    // Questo metodo scansiona la directory del programma
```

PROFILER_ MANAGER

Questa classe assembla una finestra di dialogo contenente il gestore dei profili dell'applicazione. I profili vengono conservati nella stessa directory del programma, sotto forma di file. L'accesso al loro contenuto è regolato dai metodi statici della classe Profile. Lo scopo del profile manager è mostrare la lista dei profili esistenti, in modo che l'utente possa scegliere quale profilo usare. Allo stesso tempo, il gestore dei profili fornisce anche opzioni per la creazione, la modifica e la cancellazione dei profili, appoggiandosi alla classe ProfileEditor.

```
// alla ricerca dei profili (file *.pr).
private void loadProfiles() {
    File userDir = new File(System.getProperty("user.dir"));
    // Si fa elencare i file della directory usando il filtro per
    // i file *.pr (vedi metodo accept(File f)).
    profiles = userDir.listFiles(this);
    // Legge i nomi dei file togliendo l'estensione.
    String[] profilesNames = new String[profiles.length];
    for (int i = 0; i < profiles.length; i++) {
        String name = profiles[i].getName();
        profilesNames[i] = name.substring(0,
                                        name.length() - 3); }
    // Mostra l'elenco nella JList apposita.
    profilesList.setListData(profilesNames); }
// Richiamato alla pressione di uno dei pulsanti.
public void actionPerformed(ActionEvent e) {
    Object src = e.getSource();
    if (src == button1) {
        // Seleziona profilo.
        Profile p = Profile.loadProfile(
            profiles[profilesList.getSelectedIndex()] );
        selectedProfile = p;
        dispose(); } else if (src == button2) {
        // Modifica profilo esistente.
        // Richiama il profile editor.
        ProfileEditor pe = new ProfileEditor(
            this, profiles[profilesList.getSelectedIndex()] );
        pe.show();
    } else if (src == button3) {
        // Cancella profilo.
        // Chiede conferma.
        int c = JOptionPane.showConfirmDialog(
            this, "Cancellare il profilo selezionato?", "Conferma",
                        JOptionPane.YES_NO_OPTION );
        if (c == JOptionPane.YES_OPTION) {
            // Tenta la cancellazione. In caso di fallimento
            // informa l'utente. In caso di successo aggiorna
            // la lista dei profili visualizzati.
            if (!profiles[profilesList.getSelectedIndex()
                                        ].delete()) {
             JOptionPane.showMessageDialog(this,
                 "Impossibile eliminare il profilo selezionato",
                   "Errore", JOptionPane.ERROR_MESSAGE);
            } else loadProfiles(); }
    } else if (src == button4) {
        // Crea nuovo profilo.
        // Richiama il profile editor.
        ProfileEditor pe = new ProfileEditor(this);
        pe.show();
        if (!pe.isCancelled()) loadProfiles();
    } else if (src == button5) {
        // Annulla.
        dispose(); } }
// Richiamato al cambio di selezione nella JList dei profili.
public void valueChanged(ListSelectionEvent e) {
    // I primi tre pulsanti devono essere attivi solo se c'è
    // selezione in corso.
    button1.setEnabled(!profilesList.isSelectionEmpty());
    button2.setEnabled(!profilesList.isSelectionEmpty());
    button3.setEnabled(!profilesList.isSelectionEmpty()); }
// Metodo richiesto dall'interfaccia FileFilter. Serve per
// stabilire un filtro che accetti solo i file *.pr. Viene usato
// all'interno di loadProfiles() per ottenere facilmente un
// elenco di file già scremati dalle ridondanze.
public boolean accept(File f) {
    if (f.isDirectory()) return false;
    if (f.getName().toLowerCase().endsWith(".pr")) return true;
    return false; }
// Restituisce il profilo selezionato dall'utente, che è null
// nel caso l'operazione sia stata annullata oppure se il
// profilo selezionato non è valido.
public Profile getSelectedProfile() {
    return selectedProfile;}
}
```

Fig. 1: ***Il gestore dei profili.***

ProfileManager si presenta come in Fig. 1. Mostreremo questo componente all'avvio del nostro client di posta elettronica, in modo che l'utente possa scegliere da principio quale profilo impiegare. Inoltre sarà sempre possibile richiamarlo durante l'utilizzo del programma, per passare "on the fly" da un profilo ad un altro. *ProfileManager* fa uso della classe *Profile* per caricare e salvare su file system i profili utente. Al suo interno vengono stabilite regole fondamentali del software. Anzitutto, i profili saranno conservati in file con estensione *.pr*, posizionati nella stessa directory che contiene il punto di avvio del programma. Questa scelta è stata fatta perché si intende realizzare un client di posta elettronica flessibile e leggero, che possa essere conservato in un dischetto ed usato in ogni postazione senza problemi (Java ha questa caratteristica, perché non sfruttarla?). Quindi, per il raggiungimento dello scopo, è necessario che i profili possano essere facilmente trasportati insieme al programma stesso. Da qui la scelta di tenerli nella stessa directory dell'applicazione. Quando *ProfileManager* viene avviato, il codice esegue la scansione dei file *.pr* presenti nella directory di lavoro, per poi mostrarne i nomi all'interno di una *JList* posizionata al centro della finestra. L'utente, così, potrà scegliere il profilo che intende usare e confermare la selezione con il tasto *"Seleziona"*. Il codice chiamante potrà ottenere il profilo scelto chiamando il metodo *getSelectedProfile()*.

ProfileManagerpermette anche la creazione di un nuovo profilo e la modifica o la cancellazione di quelli già esistenti. La cancellazione è davvero semplice: basta eliminare il file *.pr* associato ed aggiornare la *JList* centrale. *Creazione* e *modifica*, invece, si basano su un componente a parte, chiamato *ProfileEditor.*

LA CLASSE PROFILEEDITOR

ProfileEditor, come intuibile, realizza un'interfaccia per la creazione e la modifica dei profili:

```
// ProfileEditor.java
// Lista degli import dalla libreria di Java.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.io.*;
// Questa classe realizza una maschera utile per creare un
// nuovo profilo o per editarne uno già esistente.
class ProfileEditor extends JDialog implements ActionListener {
  // Componenti Swing.
  private JTextField[] textFields = new JTextField[5];
  private JButton button1 = new JButton("Fatto");
  private JButton button2 = new JButton("Annulla");
  // Il file associato al profilo (può essere null).
  private File profileFile;
  // Flag di cancellazione, per capire se l'utente ha
  // confermato o annullato le proprie operazioni.
  private boolean cancelled = true;
  // Costruttore a due argomenti, da richiamare per editare
  // un profilo già esistente. L'editor carica il profilo
  // specificato e permette la modifica dei campi in esso
  // contenuti.
  public ProfileEditor(ProfileManager pm, File f) {
    // Chiamata al costruttore della classe base.
    super(pm, "Editor di profili", true);
    // Registra il file del profilo da editare (o null se si tratta
    // della creazione di un nuovo profilo).
    profileFile = f;
    // Il pulsante di conferma è quello predefinito.
    getRootPane().setDefaultButton(button1);
    // Aggiunge l'ActionListener ai pulsanti della finestra.
    button1.addActionListener(this);
    button2.addActionListener(this);
    // Imposta le dimensioni dei campi di input testuali.
    for (int i = 0; i <textFields.length; i++) {
      if (i != 3) textFields[i] = new JTextField();
      else textFields[i] = new JPasswordField();
          textFields[i].setPreferredSize(new Dimension(
             250, textFields[i].getPreferredSize().height));
    }
    // Assembla l'interfaccia.
    ...
  // Costruttore con un solo argomento, da usare per
  // creare un nuovo profilo.
  public ProfileEditor(ProfileManager pm) {
    // Si appella al costruttore con due argomenti.
    this(pm, null); }
  // Ridefinizione del metodo show() di JDialog. Prima di
  // mostrare il componente carica il profilo da modificare
  // (se ce ne è uno).
  public void show() {
    // Se il profilo è diverso da null, prova a caricarlo.
    if (profileFile != null) {
      Profile p = Profile.loadProfile(profileFile);
      if (p == null) {
        // Se p è null, non è stato possibile caricare il profilo.
        JOptionPane.showMessageDialog(this, "Errore di lettura.
          " + "Impossibile leggere il profilo selezionato",
              "Errore", JOptionPane.ERROR_MESSAGE);
        dispose();
        return;
        }
        else {
          // Se il profilo è stato caricato correttamente, ne
          // mostra i campi in modo che possano essere
          // editati.
          textFields[0].setText(p.getName());
          textFields[1].setText(p.getPOP3());
          textFields[2].setText(p.getUsername());
          textFields[3].setText(p.getPassword());
          textFields[4].setText(p.getSMTP());}
      }
      // Chiama la definizione di show() contenuta nella
      // classe base.
      super.show();
  }
  // Metodo richiesto dall'interfaccia ActionListener.
  // Richiamato alla pressione dei pulsanti posizionati nella
  // finestra.
  public void actionPerformed(ActionEvent e) {
    Object src = e.getSource();
    if (src == button1) {
      // Conferma.
      if (profileFile == null) {
          // Se il profilo è nuovo, chiede all'utente di dargli
          // un nome.
          String name = JOptionPane.showInputDialog(
            this, "Nome del nuovo profilo:", "Nome profilo",
                 JOptionPane.QUESTION_MESSAGE);
          if (name == null) return;
          // In base al nuovo nome, definisce la proprietà
          // profileFile.
          profileFile = new File(System.getProperty("user.dir"),
                                          name + ".pr");
      }
      // Tenta il salvataggio del profilo.
      boolean done = Profile.saveProfile(profileFile,
                        textFields[1].getText().trim(),
                        textFields[4].getText().trim(),
                        textFields[2].getText().trim(),
```

Alcuni link consigliati sono:

http://www.csita.unige.it/servizi/posta/laposta.html

http://www.wowarea.com/italiano/aiuto/abmailit.htm

http://lagash.dft.unipa.it/AL/al263.htm

Potete trovare un buon tutorial di base sull'utilizzo delle API JavaMail all'indirizzo:

http://developer.java.sun.com/developer/onlineTraining/JavaMail/

Un bel po' di risposte alle domande più frequenti su JavaMail le trovate su jGuru, alla pagina:

http://www.jguru.com/faq/JavaMail

```
                                    textFields[3].getText(),
                                textFields[0].getText().trim());
        if (done) {
            // E' andato tutto bene. L'operazione è riuscita e
            // l'utente non l'ha annullata.
            cancelled = false;
            // Chiude e termina la finestra.
            dispose();
        } else {
            // Mostra un messaggio di errore.
            JOptionPane.showMessageDialog(this, "Errore di
                    scrittura. " + "Impossibile salvare il profilo.",
                        "Errore", JOptionPane.ERROR_MESSAGE);
        }
    } else if (src == button2) {
        // Annulla.
        dispose();
    }
  }
  // Consente l'accesso in sola lettura alla proprietà cancelled.
  public boolean isCancelled() { return cancelled; }
}
```

Anche *ProfileEditor*, proprio come *ProfileManager*, è una finestra di dialogo modale. La classe dispone di due costruttori, il primo con due argomenti, il secondo con uno soltanto. Il costruttore a due argomenti va impiegato per richiamare *ProfileManager* nel caso si desideri editare un profilo esistente: il secondo argomento richiesto, infatti, è proprio il riferimento al file che contiene il profilo da modificare. Il secondo costruttore, che non richiede alcun file, va usato per invocare *ProfileManager* nelle sue funzioni di creatore di nuovi profili. L'interfaccia comprende i campi utili per editare tutte le proprietà di un profilo.
Alla pressione del bottone di conferma, *ProfileManager* riflette le modifiche sul file system e ritorna al codice chiamante. L'aspetto finale del componente è quello mostrato in Fig. 2.

Fig. 2: *L'editor dei profili.*

Per contattare l'autore di questo articolo scrivi a carlo.pelliccia@ioprogrammo.it

PER PROVARE LE TRE CLASSI REALIZZATE...

È naturale che adesso di desideri sperimentare le funzionalità di *Profile*, *ProfileManager* e *ProfileEditor*. Siccome non sono classi indipendenti, è necessario realizzare una semplice struttura che consenta il test:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class Test extends JFrame {
  public Test() {
    super("TEST");
    setSize(200, 200);
    Dimension screen = Toolkit.getDefaultToolkit(
                                    ).getScreenSize();
    Dimension frame = getSize();
    setLocation((screen.width - frame.width) / 2,
                    (screen.height - frame.height) / 2 );
    setDefaultCloseOperation(EXIT_ON_CLOSE);
  }
  public static void main(String[] args) {
    Test t = new Test();
    ProfileManager pm = new ProfileManager(t);
    t.show();
    pm.show();
    Profile p = pm.getSelectedProfile();
    if (p == null) {
      System.out.println("Non hai selezionato alcun profilo.");
    }
    else {
      System.out.println("Nome visualizzato: " +
                                        p.getName());
      System.out.println("POP3: " + p.getPOP3());
      System.out.println("POP3 username: " +
                                    p.getUsername());
      System.out.println("POP3 password: " +
                                    p.getPassword());
      System.out.println("SMTP: " + p.getSMTP());
    }
  }
}
```

Avviate la classe *Test* per sperimentare i risultati ottenuti con questa parte del tutorial.

IL MESE PROSSIMO...

Il mese prossimo torneremo ad impugnare le API JavaMail per imbastire lo scheletro funzionale del nostro client di posta elettronica, sfruttando inoltre il meccanismo di gestione dei profili utente illustrato nelle pagine che avete appena letto.

Carlo Pelliccia

RDBMS to ODBMS: i database per applicazioni object-oriented

Oggetti e persistenza

In questa serie di articoli vedremo come gestire efficacemente la persistenza degli oggetti, partendo dai tradizionali RDBMS fino ad arrivare ai più innovativi ODBMS.

Qualsiasi applicazione ha bisogno di accedere in lettura o scrittura a dati persistenti. Nel mondo informatico la persistenza è dominata in gran parte dalla tecnologia dei database relazionali (RDBMS – Relational database management system). Con l'avvento del paradigma object-oriented, i linguaggi di programmazioni hanno acquistato maggiore successo mentre i dati spesso continuano ad essere utilizzati secondo la struttura relazionale. In questo articolo e nei successivi vedremo le possibili integrazioni tra applicazioni object-oriented e persistenza fino ad arrivare ad esaminare una nuova tecnologia: i database ad oggetti (ODBMS – Object database management system). Arriveremo a mostrare gradualmente l'architettura e le caratteristiche di tale tipologia di database, mostrandone i vantaggi e gli svantaggi rispetto all'approccio tradizionale. Partiremo, pertanto, evidenziando alcuni limiti dei database relazionali che si riscontrano durante lo sviluppo di un'applicazione object-oriented e le soluzioni possibili con cui questi possono essere superati. Passando per altre soluzioni proposte sul mercato, successivamente introdurremo i database ad oggetti e la loro architettura e mostreremo come tale tecnologia rappresenti il traguardo naturale di un'applicazione object-oriented.

PROGRAMMAZIONE OBJECT-ORIENTED

Essa rappresenta ormai una metodologia collaudata e matura. Parliamo di metodologia poiché, accanto ai linguaggi di programmazione classici o più recenti (SmallTalk, C++, Java, C#), sono disponibili tecniche e linguaggi di modellazione (ad esempio l'UML) per l'analisi ed il design di applicazioni. Nel corso dell'articolo, daremo per scontato che siano noti i principi dell'object-orientation. Riportiamo in ogni caso di seguito i principali vantaggi:

- velocità di sviluppo del codice
- facilità di manutenzione
- riutilizzo del codice

Nello sviluppo di un'applicazione, si arriva alla fine a dover affrontare il problema della persistenza: ossia leggere e/o scrivere dati da uno storage. Poiché la metodologia utilizzata è quella object-oriented, i dati in questione in realtà sono oggetti. La problematica si riduce quindi alla persistenza di tali oggetti sui possibili storage.

RDBMS

Nell'esaminare i possibili storage, iniziamo dai database relazionali. Essi hanno il grosso vantaggio di essere una tecnologia ampiamente collaudata e popolare. Sono disponibili sul mercato diversi prodotti, le cui prestazioni sono notevoli, e soprattutto esiste una letteratura ampia ed esauriente che offre soluzioni a qualsiasi problematica. Purtroppo nell'utilizzo con la programmazione object-oriented si riscontrano alcune limitazioni, che possono in parte essere superate grazie ad alcune soluzioni che esamineremo.

DA CLASSI A TABELLE

Rispetto alla programmazione tradizionale, in cui la base di partenza è rappresentata dal modello dei dati, espresso da un diagramma ER (entità-relazioni), nella programmazione object-oriented il punto iniziale è rappresentato da un diagramma delle clas-

UML

L'acronimo UML sta per Unified Modeling Language. Esso è un linguaggio di modellazione, opera dell'OMG (Object Management Group), che permette di specificare, visualizzare e documentare modelli di sistemi software. Rappresenta uno standard per l'analisi e la progettazione di applicazioni object-oriented, avvalendosi di un grande numero di tool presenti sul mercato.

si. Queste ultime sono allo stesso tempo un modello di dati e di comportamenti, poiché modellano un processo di business del mondo reale. Una volta che, quindi, il processo di analisi ha individuato un gruppo di classi, tra loro legate da relazioni di diverso tipo, rappresentanti le entità di business, si ha a disposizione un diagramma delle classi. Di seguito riportiamo le possibili relazioni che possono verificarsi tra le istanze delle classi:

- **ereditarietà**
- **associazione**
- **aggregazione**

Nella maggior parte dei casi, tali classi di business necessitano di essere rese persistenti ma per far ciò bisogna in qualche modo riportarle in un diagramma ER. In pratica ciò di cui abbiamo bisogno è di mappare una classe ad una o più tabelle del database relazionale. Si potrebbe pensare che tale operazione sia banale, poiché basta creare, per ogni classe, una tabella le cui colonne sono relative agli attributi persistenti ed in cui ogni riga memorizza un diverso oggetto. Già a questo punto, va notato che ci sono dei dettagli che richiedono una certa attenzione, poiché occorre mappare anche ogni tipo di dato di un attributo nel corrispondente tipo di dato per la colonna. Non ci sono problemi finché si tratta di tipi semplici, come interi, stringhe, ecc.. Ma non appena subentrano tipi complessi, quali liste, array, classi, l'operazione si complica.

OBJECT-RELATIONAL-BRIDGE

È un tool, per il mapping classi-tabelle, completamente open source e facente parte del progetto Apache DB http://db.apache.org/ Conforme allo standard ODMG 3.0, aderisce alle nuove specifiche JDO (Java Database Objects) della Sun.

EREDITARIETÀ

Osserviamo più in dettaglio un problema classico del mapping, quello relativo alla relazione più importante dell'object-oriented: l'ereditarietà. Nella letteratura si trova molto materiale al riguardo e le soluzioni proposte sono ormai sufficientemente collaudate. Come esempio, basiamoci su un'applicazione di anagrafica di clienti ed impiegati di un'azienda in cui è presente una classe astratta *Persona*

GLOSSARIO

ORACLE TOPLINK

Oracle TopLink è uno strumento, disponibile all'interno dell'offerta Oracle, per il mapping classi-tabelle. Si compone di un tool grafico in cui specificare le regole di mapping ed impostarne le opzioni e di una libreria da utilizzare all'interno dell'applicazione Java.

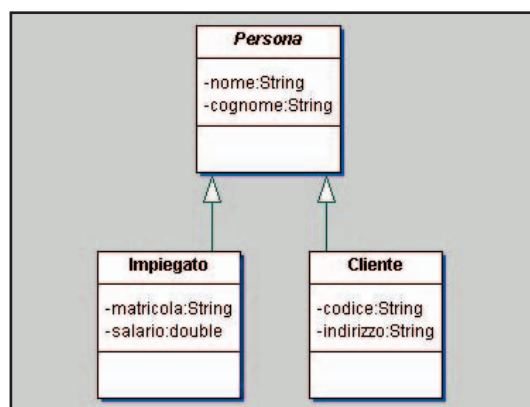



Fig. 1: ***Anagrafica dei clienti ed impiegati di un'azienda.***

da cui derivano le classi *Cliente* e *Impiegato*. Ognuna di esse, come possiamo osservare nel class diagram mostrato in Fig. 1, definisce degli attributi specifici. Per implementare tale struttura gerarchica su un database relazionale abbiamo a disposizione tre differenti strategie di mapping:

- **verticale**
- **orizzontale**
- **filtrato**

Il primo caso è molto semplice: ogni classe, concreta o astratta, è mappata in una propria tabella. Le associazioni alle eventuali classi padre vengono implementate mediante l'uso di foreign key e primary key sulle relative tabelle. In tal modo occorre effettuare delle query di join per instanziare una classe concreta. In Fig. 2 sono mostrate le tabelle necessarie relativamente al class diagram mostrato in precedenza. Si può notare che le tabelle *Impiegato* e *Cliente* sono in relazione con la tabella *Persona* mediante la colonna *personaid*.

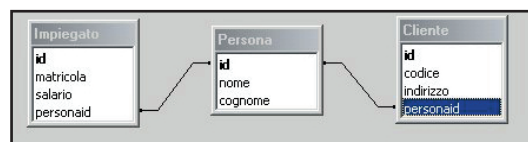



Fig. 2: ***Usando il mapping verticale ogni classe ha la propria tabella.***

Tale soluzione crea uno schema relazionale molto simile alla struttura gerarchica delle classi da mappare ma, nel caso di catene di ereditarietà molto profonde, non offre buone prestazioni. In tal caso, infatti, per caricare un oggetto di una classe in fondo alla catena, sarebbe necessario navigare su molte tabelle (tante quante sono le classi da cui questa deriva) aumentando così il tempo richiesto dalla query. Con il mapping orizzontale, viceversa, tale limitazione è superata, poiché solo ogni classe concreta viene mappata dalla propria tabella. In tal modo si riduce il numero delle tabelle, poiché le classi astratte non vengono considerate. Che fine fanno i loro attributi allora? Essi vengono "riportati" sulle rispettive classi concrete e di conseguenza mappati nelle corrispondenti tabelle. Così facendo si appiattisce la struttura gerarchica poiché si eliminano le classi astratte. Nel caso del nostro esempio, ciò dà luogo a due sole tabelle, come mostrato in Fig. 3. Notiamo che non

Fig. 3: ***Il mapping orizzontale richiede un minor numero di tabelle.***

esistono più relazioni tra le tabelle. In effetti, esse sono scomparse del tutto, poiché il nostro class diagram è molto semplice, non esistendo classi concrete che derivano le une dalle altre. Considerando una struttura più articolata, come quella mostrata in Fig. 4, dove sono definite due nuove classi Tecnico ed Amministrativo, otterremo che le rispettive tabelle saranno in relazione con quella della classe padre Impiegato.

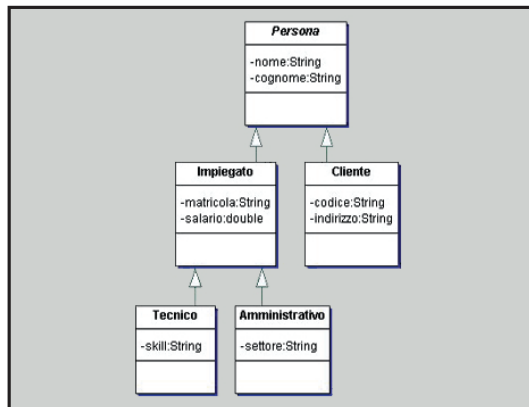



Fig. 4: *Le classi Tecnico ed Amministrativo specializzano ulteriormente la classe Impiegato.*

Tale soluzione, pertanto, mantenendo un design semplice, richiede un minor numero di tabelle e conseguentemente di relazioni tra loro, ottimizzando così il tempo richiesto dalle query ed offrendo buone performance. D'altra parte va notato che se un attributo di una classe astratta cambia, occorre modificare tutte le tabelle relative alle classi concrete derivanti da questa. Pertanto, nel caso di strutture gerarchiche in cui sono presenti molte classi concrete e queste derivano la maggior parte degli da classi astratte, tale soluzione non offre facilità di manutenzione ed adattabilità a modifiche. Vediamo infine il mapping filtrato. Tale soluzione utilizza una sola tabella per tutte le classi della struttura gerarchica. La singola tabella conterrà tante colonne quanti sono gli attributi di tutte le classi (astratte e concrete), appiattendo completamente la struttura. Per distinguere, inoltre, tra le righe della tabella il tipo di istanza, è necessario introdurre una colonna di filtraggio, il cui valore identifica in qualche modo la classe concreta. In questo caso, trattandosi di una

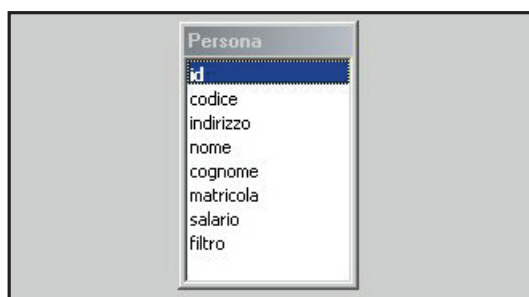



Fig. 5: *Nel mapping filtrato l'intera gerarchia è modellata da una sola tabella.*

sola tabella, le query non richiedono operazioni di join, offrendo in tal modo ottime performance. Tale soluzione, viceversa, viola le regole di normalizzazione, causando potenzialmente un alto numero di colonne con valori nulli ed aumentando lo spazio perso. Essa pertanto si rivela efficace se la maggior parte degli attributi delle classi sono ereditati da classi astratte.

ASSOCIAZIONI ED AGGREGAZIONI

Dopo l'ereditarietà, altre relazioni sono possibili tra gli oggetti. Diamo un piccolo sguardo ad esse prima di vedere come implementarle in un database relazionale. La relazione più generale è l'associazione, la quale rappresenta l'abilità di un oggetto di inviare messaggi ad un altro (ciò tipicamente è realizzato tramite un riferimento o puntatore all'istanza da invocare). L'aggregazione, viceversa, è un'associazione, che permette di modellare la relazione esistente tra un'entità e le sue parti, in cui non esistono relazioni cicliche (ossia una parte non può contenere la sua entità). Un esempio è il caso dell'aeroplano costituito nelle sue componenti dalle ali, la fusoliera, ecc… Quando andiamo a riportare tali relazioni su un database relazionale, in realtà ci si riduce a considerare tre tipologie, ognuna distinta in base alla cardinalità: *uno ad uno; uno a molti* e *molti a molti* Riprendendo l'esempio precedente, aggiungiamo le classi che modellano il processo degli ordini di libri da parte di un cliente. La relazione uno ad uno, esistente tra cliente e ordine, è facilmente realizzabile, inserendo, nella tabella *Ordine*, una colonna foreign key, contenente il valore della primary key della riga della tabella *Cliente* da referenziare. Anche la relazione uno a molti può essere implementata mediante una colonna foreign key da inserire nella tabella relativa alla classe con cardinalità maggiore (in tal caso la tabella *Posizione*). La relazione molti a molti è infine quella più complessa, ed è comunemente implementata mediante una join table. Essa è una semplice tabella che, mediante due colonne foreign key, mette in relazione righe di due diverse tabelle, contenendo i valori delle rispettive primary key. La scelta delle diverse strategie di mapping che abbiamo visto può essere eseguita sia manualmente sia per mezzo di un tool. Mostreremo vantaggi e svantaggi di tali tool più avanti. Per ora è sufficiente comprendere che nell'integrazione tra un'applicazione object-oriented ed un database relazionale occorre considerare almeno due necessità: un'operazione di mapping da classi a tabelle nel momento in cui si progetta il database; uno strato software (detto comunemente *persistence layer*) che in modo trasparente permette all'applicazione di leggere e scrivere oggetti nel database.

SQL:1999

La maggior parte degli ORDBMS (tra cui Oracle, DB2, ecc.) aderiscono allo standard SQL:1999. Esso arricchisce il classico linguaggio SQL, dando la possibilità all'utente di definire nuovi tipi di dati, supportando l'ereditarietà singola ma non quella multipla delle interfacce (come in Java).

Fig. 6: Un ordine, composto da posizioni, è specifico di ogni cliente.

OBJECT WRAPPER ED ORDBMS

Come conseguenza di tutto ciò, molti produttori di database relazionali hanno tentato o tentano tuttora di rispondere al crescente entusiasmo del mercato verso la tecnologia object-oriented. Alcuni di essi hanno intrapreso la strada degli Object Wrapper, altri quella di introdurre la funzionalità ad oggetti nei loro database, creando soluzioni ibride quali gli ORDBMS (Object-relational database management system). Un Object Wrapper è uno strato software, posto al di sopra del database, che offre, verso l'esterno, un'interfaccia object-oriented del sottostante meccanismo relazionale. Tale soluzione risolve il problema dal punto di vista della programmazione, perché effettua automaticamente il mapping tra istanze di una classe e righe di tabelle del database relazionale, ma non permette di ottenere risultati rilevanti in termini di performance. Ciò ovviamente è dovuto alla presenza intrinseca di maggior codice da eseguire all'interno dell'applicazione. Le operazioni di mapping sono automatizzate mediante l'uso di appositi tool, i quali risultano molto efficienti nel caso di un modello dei dati già esistente, data la semplicità dei modelli relazionali. Ciò viceversa può non accadere per modelli object-oriented, in cui sono presenti classi legate tra loro in modo complesso. Riepiloghiamo di seguito le maggiori problematiche di tale soluzione. Per un modello di classi complesso, il processo di mapping può risultare molto lungo poiché occorre indicare manualmente da quale colonna della tabella leggere o scrivere un attributo della classe. Generalmente tali tool offrono differenti algoritmi di mapping, i quali lavorano inserendo il nome della classe nella riga della tabella, soluzione non particolarmente elegante. Puntatori e collezioni sono mappate in colonne e gli ID utilizzati corrispondono in genere a valori che sono codificati all'interno del codice, con conseguente difficoltà di manutenzione. Il mapping mediante foreign key richiede una grande quantità di spazio su disco. Spesso tali tool non consentono di controllare ed eliminare gli errori che possono verificarsi quando si associa manualmente un attributo ad una colonna. Il codice relativo al persistence layer introduce comunque un overhead nell'applicazione. A causa delle grossolane perfomance, spesso tali tool propongono delle ottimizzazioni che vanno ad impattare negativamente sul design, introducendo inoltre nell'applicazione una forte dipendenza con il tool utilizzato. Un ORDBMS, viceversa, rappresenta una soluzione ibrida a metà tra i tradizionali database relazionali e quelli ad oggetti. Esso si basa ancora su uno strato di mapping classi-tabelle ma in più offre altre caratteristiche quali: nuovi tipi di dati, per la gestione ad esempio di dati multimediali, serie di dati, ecc. Essi sono integrati con la sottostante struttura relazionale, ma non sono espressi secondo il paradigma object-oriented poiché non supportano completamente l'ereditarietà. Pertanto non permettono di definire nuovi complessi tipi di dati; gestione di attributi complessi, creando tabelle in cui è specificato, come attributo, il nome di un'altra. In tale modo si può gestire un'associazione tra due classi, ma tale funzionalità rappresenta solo una simulazione a livello sintattico, giacché i dati vengono memorizzati ancora su tabelle diverse ed il loro caricamento richiede sempre delle operazioni di join, richiedenti tempo. Sicuramente le performance ottenute in questo caso sono migliori ma, poiché l'ORDBMS non supporta completamente la tecnologia object-oriented e non modella in modo efficiente relazioni complesse tra i dati, esso può essere considerato come un database relazionale migliorato. In effetti, tale soluzione può apparire più rassicurante per chi lavora con un database relazionale, anche perché il nome dà l'idea di una transizione più facile verso la tecnologia object-oriented. In realtà tale soluzione può essere adatta al caso di applicazioni esistenti basate su database relazionali ed in cui i dati non hanno una grossa complessità.

CONCLUSIONI

Abbiamo visto come poter gestire la persistenza degli oggetti su una base dati relazionale e gli svantaggi che ciò comporta nel caso di design object-oriented. Tali svantaggi riguardano innanzitutto una minore facilità e velocità di sviluppo del codice, e sopratutto un degradamento delle performance nel momento in cui occorre gestire modelli di classi legate tra loro in modo complesso. Abbiamo quindi presentato anche alcune soluzioni ibride, quali gli ORDBMS, che permettono prestazioni migliori, mantenendo una sottostante struttura relazionale.
Vedremo nel prossimo articolo come tali limitazioni possono essere superate mediante un'altra tecnologia, quella dei database ad oggetti (ODBMS).

David Visicchio

L'AUTORE

David Visicchio è laureato in Ingegneria Informatica e lavora a Roma come designer/developer presso una multinazionale software, leader nel mercato per la persistenza ed il middleware di sistemi object-oriented. I suoi interessi sono orientati principalmente alle architetture distribuite basate su piattaforme J2EE e J2SE.

BIBLIOGRAFIA

- **ODBMS VS ORDBMS: WHICH IS RIGHT FOR YOU?** *Douglas Berry* (Software Development)
- **WHICH DATABASE IS RIGHT FOR THE NEW INFORMATION SYSTEMS?** (Versant Corporation) http://www.versant.com/resources/control/white_papers/WP_001001r6W.pdf
- **BUILDING OBJECT APPLICATIONS THAT WORK** *Scott Ambler* (Cambridge University Press)
- **UML Distilled** *Martin Fowler* (Addison-Wesley)
- **Design of a Persistence layer** *Scott Ambler* http://www.ambysoft.com/persistenceLayer.html

I trucchi del mestiere

Tips&Tricks

La rubrica raccoglie trucchi e piccoli pezzi di codice che solitamente non trovano posto nei manuali, ma sono frutto dell'esperienza di chi programma. Alcuni trucchi sono proposti dalla Redazione, altri provengono da una ricerca su internet, altri ancora ci giungono dai lettori. Chi vuole contribuire potrà inviarci i suoi tips&tricks preferiti che, una volta scelti, verranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

LEGGERE E SCRIVERE IN UN FILE .INI

Il tip, inserito in un modulo BAS (o CLS), permette di leggere e scrivere informazioni nei file .INI, il tutto senza utilizzare complicate procedure di ricerca all'interno di un file, ma utilizzando le API di Windows.

Tip fornito dal sig. S.Tomaselli

```
Declare Function WritePrivateProfileString Lib "Kernel32" Alias
                                        "WritePrivateProfileStringA"
ByVal lpApplicationname As String, ByVal lpKeyName As Any, ByVal
            lsString As Any, ByVal lplFilename As String) As Long
Declare Function GetPrivateProfileInt Lib "Kernel32" Alias
                                      "GetPriviteProfileIntA" (ByVal
lpApplicationname As String, ByVal lpKeyName As String, ByVal
             nDefault As Long, ByVal lpFileName As String) As Long
Declare Function GetPrivateProfileString Lib "Kernel32" Alias
                                   "GetPrivateProfileStringA" (ByVal
lpApplicationname As String, ByVal lpKeyName As String, ByVal
              lpDefault As String, ByVal lpReturnedString As String,
              ByVal nSize As Long, ByVal lpFileName As String) As
Long
Global File
Global appname
Global Keyname
Global Value
Public Function Store(File As String, Heading As String, Section As
                                         String, Value As String)
Dim lpAppName As String, lpFileName As String, lpKeyName As
                                              String, lpString As
String
Dim U As Long
lpAppName = Heading
lpKeyName = Section
lpString = Value
lpFileName = File
U = WritePrivateProfileString(lpAppName, lpKeyName, lpString,
                                                      lpFileName)
If U = 0 Then
Beep
Else
Store = "Success"
End If
End Function
Public Function GetValue(File As String, Heading As String, Section
                                                       As String)
Dim x As Long
Dim Temp As String * 50
Dim lpAppName As String, lpKeyName As String, lpDefault As String,
                                             lpFileName As String
Temp = Space$(50)
lpAppName = Heading
lpKeyName = Section
lpDefault = no
lpFileName = File
x = GetPrivateProfileString(lpAppName, lpKeyName, lpDefault,
                                   Temp, Len(Temp), lpFileName)
If x = 0 Then
Beep
Else
en = InStr(1, Temp, Chr$(0))
If en > 0 Then Temp = Mid$(Temp, 1, en - 1)
GetValue = Trim$(Temp)
End If
End Function
```

UN CONTROLLO LISTVIEW IN MODALITÀ "REPORT"

Sovente capita di realizzare delle applicazioni in cui è necessario utilizzare un cotrollo *ListView*, in modalità *Report*, per la visualizzazione dei dati. Purtroppo non è possibile editare gli stessi come la stessa semplicità di un *DataGrid*. Il tip proposto consente di "eliminare" questo problema visualizzando un controllo *ComboBox* all'interno di una colonna di una riga selezionata, quando si fa click sul controllo ListView, dando la possibilità di cambiare il valore di un item, scegliendo tra una delle voci proposte dal ComboBox visualizzato. Con la stessa tecnica è possibile aggiungere altri controlli standard come ProgressBar, Command Buttons, ecc. Trovate l'applicazione completa allegata al cd-rom che accompagna la rivista o nella sezione download del sito *www.ioprogrammo.it*

Tip fornito dal sig. P. Libro

UN RISOLUTORE D'ESPRESSIONI MATEMATICHE

Il codice riportato è un risolutore d'espressioni matematiche numeriche (non letterali e non equazioni). Rispetta le priorità delle operazioni (prima potenze e radici, poi moltiplicazioni e divisioni ed in ultimo addizioni e sottrazioni). È possibile utilizzare vari livelli di parentesi ma solo parentesi tonde. Supporta le funzioni matematiche standard di Visual Basic. È progettato per poter funzionare con il separatore decimale ma anche con la virgola. La funzione *Simple* risolve le espressioni senza parentesi. La funzione principale *Solve* estrae il contenuto delle parentesi e lo fa risolvere alla funzione *Simple*. Impostando *ShowPassages* su *True* saranno restituiti i passaggi compiuti dalla funzione per risolvere l'espressione. Trovate le funzioni, definite in modulo *.BAS*, allegato al cd-rom che accompagna la rivista o nella sezione download del sito *www.ioprogrammo.it*

Tip fornito dal sig. S. Tomaselli

UNA COPIA PERSONALIZZABILE

Il codice in allegato è relativo ad un programma scritto da me. Il codice permette di copiare dei file da una cartella di origine in una cartella di destinazione, dando la possibilità sia di scegliere il tipo di file da copiare ((word, excel, txt,rtf, o tutti,...) trasformandone inoltre il nome_file. estensione in datacreazione_nomefile_numero progressivo.estensione.

Tip fornito dal Sig. C.Calabrò

```
Dim pattern As String
Private Sub cmdtrad_Click()
Dim SourceFile, DestinationFile
Dim count As Integer
Dim fs, f, s
Set fs = CreateObject("Scripting.FileSystemObject")
count = 0
For count = 1 To File1.ListCount
SourceFile = Dir1.Path & "\" & File1.List(count - 1) ' Definisce il
                                              nome del file di origine.
stringaSE = Left(File1.List(count - 1), (Len(SourceFile) -
                          (Len(Dir1.Path) + 5))) 'stringa senza estensione
stringaCE = Left(File1.List(count - 1), (Len(SourceFile) -
                              (Len(Dir1.Path)))) 'stringa con estensione
  Set f = fs.GetFile(SourceFile)
  data_creazione = f.DateCreated
  giorno_creazione = Left(data_creazione, 2)
  mese_creazione = Right(Left(data_creazione, 5), 2)
  anno_creazione = Right(Left(data_creazione, 10), 4)
```

IL TIP DEL MESE

GLI UTENTI CAMBIANO LA DIMENSIONE DEI CONTROLLI

Grazie a due semplice chiamate ad API, è possibile mettere gli utenti in grado di ridimensionare i controlli presenti nelle applicazioni che realizziamo, in modo del tutto simile a quanto si fa nel Design Mode di Visual Basic 6: cliccando su un bordo, o su uno spigolo, e trascinandolo per ridefinire i contorni del controllo. Il codice suppone che nella form principale sia presente una picture box (Picture1) e gestisce il caso del ridimensionamento orizzontale, ma è facilmente estensibile a tutti gli altri casi.

Tip fornito dal sig.R.Fabozzi

```
Private Declare Function ReleaseCapture Lib _
  "user32" () As Long
Private Declare Function SendMessage Lib _
  "user32" Alias "SendMessageA" (ByVal hWnd _
  As Long, ByVal wMsg As Long, ByVal wParam _
  As Long, lParam As Any) As Long
Private Const WM_NCLBUTTONDOWN = &HA1
Private Const HTLEFT = 10
Private Const HTRIGHT = 11
Private Sub Picture1_MouseDown(Button As _
  Integer, Shift As Integer, X As Single, Y As Single)
  Dim nParam As Long
  With Picture1
    If (X > 0 And X < 100) Then
      nParam = HTLEFT
    ElseIf (X > .Width - 100 And X < .Width) Then
    ' these too
      nParam = HTRIGHT
    End If
    If nParam Then
      Call ReleaseCapture
      Call SendMessage(.hWnd, _
        WM_NCLBUTTONDOWN, nParam, 0)
    End If
  End With
End Sub
Private Sub Picture1_MouseMove(Button As _
  Integer, Shift As Integer, X As Single, Y As Single)
  Dim NewPointer As MousePointerConstants
  If (X > 0 And X < 100) Then
    NewPointer = vbSizeWE
  ElseIf (X > Picture1.Width - 100 And X < _
    Picture1.Width) Then
    NewPointer = vbSizeWE
  Else
    NewPointer = vbDefault
  End If
  If NewPointer <> Picture1.MousePointer Then
    Picture1.MousePointer = NewPointer
  End If
End Sub
```

```
    estensione = Right(SourceFile, 4)
stringaSECD = anno_creazione & mese_creazione &
    giorno_creazione & "_" & stringaSE & "_" & count 'Stringa Senza
                                              Estensione Con Data
    stringaCE = stringaSECD & estensione
    DestinationFile = Dir2.Path & "\" & stringaCE  ' Definisce il nome
                                              del file di destinazione.
    FileCopy SourceFile, DestinationFile   ' Copia il file di origine sul
                                              file di destinazione.
    File2.Refresh
Next
    If (count - 1 > 1) Then
        MsgBox ("Sono stati copiati " & File2.ListCount & " files"), , App.Title
    Else
        If (count - 1 = 1) Then
            MsgBox ("E' stato copiato un solo file!"), , App.Title
        Else
            MsgBox ("Nessun file copiato!"), , App.Title
        End If
    End If
End Sub
Private Sub Dir1_Change()
    File1.Path = Dir1.Path
End Sub
Private Sub Dir2_Change()
    File2.Path = Dir2.Path
End Sub
Private Sub Drive1_Change()
    On Error GoTo error
    Dir1.Path = Drive1.Drive
error: If error = "Periferica non disponibile" Then MsgBox ("Drive vuoto")
End Sub
Private Sub Drive2_Change()
    On Error GoTo error
    Dir2.Path = Drive2.Drive
error: If error = "Periferica non disponibile" Then MsgBox ("Drive vuoto")
End Sub
Private Sub File2_DblClick()
    File2.Refresh
End Sub
Private Sub Form_Load()
    cmdtrad.Caption = "Traduci >>"
    cmdesc.Caption = "Esci"
    Frame1.Caption = " Sorgente "
    Frame2.Caption = " Destinazione "
    Form1.Caption = App.Title
    With lbltitle
        .Caption = App.Title & " 1.0"
        .Font.Name = "times"
        .FontSize = 32
        .FontBold = True
        .FontItalic = True
        .ForeColor = "9772354"
        .Alignment = 2
    End With
End Sub
Private Sub Form_Unload(Cancel As Integer)
    MsgBox ("Ricordati di cancellare questo msgbox, di aumentare il
                                              timer e di gestire gli errori")
End Sub
Private Sub Option1_Click(Index As Integer)
    Select Case (Index)
        Case 0: pattern = "*.doc"
        Case 1: pattern = "*.xls"
        Case 2: pattern = "*.mdb"
        Case 3: pattern = "*.txt"
        Case 4: pattern = "*.rtf"
        Case 5: pattern = "*.*"
    End Select
    File1.pattern = pattern
    File2.pattern = pattern
    File1.Refresh
End Sub
```

COMPILARE LE CLASSI "AL VOLO"

Il tip proposto riguarda la compilazione a run-time, di classi java in modo veloce ed efficiente usando una classe delle librerie del java2 sdk in alternativa al metodo, più intuitivo ma meno elegante, che prevede la creazione di un processo separato e l' invocazione mediante la *exec* del compilatore javac. Un'applicazione può, quindi, creare i propri file java e ottenerne una compilazione per poi caricane le classi appena create tramite la *Reflection*; ciò garantisce la disponibilità di classi da essa stessa create.
La libreria da includere è la *\lib\tools.jar* presente nella directory del Java 2 SDK.

Tip fornito dal sig. M.Pace

```
import java.io.*;
import java.util.*;
public class Compilertip {
  public static void main(String args[]) throws IOException  {
      if(args.length==0){
         System.out.println("Linea di comando: java Compilertip file1
                                                        file2 ecc");
      }else{Compile(args);}
   } /*Compila i file i cui nomi sono nel vettore fornito come parametro*/
   static public boolean Compile(String[] filenames){ try{
        int filecompilati = 0;
        for(int i=0;i<filenames.length;i++){
          String fname = filenames[i];
          String sourceFile =  this.currentpath+""+fname[i] +".java";
          int compileReturnCode = com.sun.tools.javac.Main.compile(
                                         new String[] {sourceFile});
          if (compileReturnCode == 0) {
            System.out.println("File "+ filenames[i] + " compilato
                                                   con successo.");
            filecompilati++; }else{
            System.out.println("Errore nella compilazione di "+
                                        filenames[i] + ". Termino.");
            return false; }
```

```
        }//for
        if(filecompilati == filenames.length){
          System.out.println("Tutti i file ("+filecompilati+")
                                            compilati con successo.");
          return true; }
        catch(Exception e){
        /*... gestione dell' errore...*/ }
        return false; }   }
/*
    javac -classpath  c:\j2sdk1.4.2\lib\tools.jar Compilertip.java
    java  -classpath  c:\j2sdk1.4.2\lib\tools.jar Compilertip.java
*/
```

INPUT BUFFERIZZATO

Un tip Java che risolve un problema spesso riscontrato, quello della gestione dell'input da tastiera non bufferizzato e con echo disabilitato (utile anche per l'inserimento di password). A questo scopo faremo uso dei JNI (Java Native Interface) di Java, che offre al programmatore la possibilità di far interagire codice C/C++ con il codice Java in maniera relativamente semplice.

Tip fornito dal sig. C. Sicilia

```
package it.kya.io;
import java.io.IOException;
public class KeyBoard
{           static
            { System.loadLibrary("KeyBoard"); }
            public static native int read() throws IOException; }
```

La classe *KeyBoard* fornisce l'interfaccia necessaria al nostro scopo, dichiarando il metodo read. Il secondo passo è quello di compilare la classe con il semplice comando:

```
javac KeyBoard.java
```

La compilazione creerà la classe *KeyBoard.class* nel package *it.kya.io*. Il terzo passo è quello di creare l'header per le funzioni C da implementare, a questo scopo useremo il tool javah, presente nella cartella bin del jdk:

```
javah –jni –o KeyBoard.h it.kya.io.KeyBoard
```

Questa procedura genererà il file *KeyBoard.h* del tutto simile a quanto proposto:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class it_kya_io_KeyBoard */
#ifndef _Included_it_kya_io_KeyBoard
#define _Included_it_kya_io_KeyBoard
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     it_kya_io_KeyBoard
 * Method:    read
 * Signature: ()I
 */
JNIEXPORT jint JNICALL Java_it_kya_io_KeyBoard_read
(JNIEnv *, jclass);
#ifdef __cplusplus
 }
#endif
#endif
```

A questo punto non ci rimane che implementare le funzioni in codice C, il prossimo passo sarà, quindi, quello di creare un file *KeyBoard.c* contenente le funzioni dichiarate nell'header. Bisogna dire che una funzione scritta in C non è portabile e richiede quindi che venga ricompilata sui diversi sistemi operativi e visto che la portabilità è un punto di forza di Java bisognerà, quantomeno, fornire una libreria per Windows e una per Linux. Essendo L'I/O generalmente legato al sistema operativo, in particolare la lettura dalla tastiera, avremo bisogno di librerie diverse per i due sistemi. Per Windows useremo *conio.h,* per Linux *curses.h,* utilizzeremo le macro per gestire una compilazione sul medesimo file, quindi nel file *KeyBoard.c* scriveremo:

```
#if defined _WIN32
#           include <conio.h>
#elif defined __linux__
#           include <curses.h>
#else
#           error Operating system is undefined or unknown!
#endif
#include <jni.h>
#include "KeyBoard.h"
JNIEXPORT jint JNICALL Java_it_kya_io_KeyBoard_read
   (JNIEnv *env, jobject obj) {
   jint ch;
#if defined _WIN32
   ch = _getch();
#elif defined __linux__
   initscr();
   cbreak();
   noecho();
   ch = getchar();
   echo();
    endwin();
   if(ch == ERR) {
      jclass newExc = (*env)->FindClass(env, "IOException");
      if (newExc != NULL)
      { (*env)->ThrowNew(env, newExc, "I/O Input error");}
      (*env)->DeleteLocalRef(env, newExc);
    }
   #endif
   return ch;
}
```

LA CONVALIDA DELLA PARTITA IVA

Programmando in ambito web (JSP ecc.), ci si trova spesso a dover controllare la correttezza di alcuni campi, inseriti dall'utente. Ciò, spesso, porta alla perdita di non poco tempo, per lo sviluppo di funzioni di controllo specializzate per il campo. In genere i più laboriosi sono il codice fiscale (di cui una soluzione è

già stata pubblicata) e la Partita Iva. La funzione qui riportata esegue un controllo su una stringa contenete una Partita Iva e restituisce valore true se essa è valida, false altrimenti.

Tip fornito dal sig. G. Sanfilippo

```
boolean checkValidPartitaIva(String PartitaIva)
{ int Somma01 = 0; int Somma02 = 0; int CheckNumber;
 if (PartitaIva.length() != 11) return false;
 try
 { if(Float.parseFloat(PartitaIva) < Float.parseFloat("0")) return false;
 }  catch (NumberFormatException NFE) {NFE.printStackTrace();
                                        return false;}
 for (int i = 0; i < 9; i += 2) {
   CheckNumber = Integer.parseInt(""+PartitaIva.charAt(i));
   Somma01 += CheckNumber;
   CheckNumber = Integer.parseInt(""+PartitaIva.charAt(i+1));
   Somma01 += Math.floor(CheckNumber/5) + (CheckNumber
                                        << 1) % 10; }
 Somma02 = 10 - (Somma01 % 10);
 CheckNumber = Integer.parseInt(""+PartitaIva.charAt(10));
 if (Somma02 != CheckNumber) return false; else return true;
}
```

BYPASSARE LA CONSOLE DI WINDOWS

Questo semplice tip illustra come attraverso l'istruzione "pragma comment" sia possibile bypassare la console di Windows.

Tip fornito dal sig. E. Di Santo

```
//---Console "bypass" – E. Di Santo ---//
//Queste istruzioni sono necessarie
//per poter utilizzare la funzione "PlaySound"
#include <windows.h>
#pragma comment(lib,"winmm.lib")
//Questa è l'istruzione vera e propria
//che permette di "bypassare" la console
#pragma comment( linker, "/subsystem:\"windows\"
                              /entry:\"mainCRTStartup\"" )
void main()
{
  PlaySound("SystemStart",NULL, SND_ALIAS|SND_SYNC);
  return;
}
```

CREAZIONE DI COMPONENTI A RUN-TIME

Il tip consente di creare componenti a run-time.

Tip fornito dal sig. L.Nalli

La prima operazione da compiere consiste nella preparazione della windows: si inseriscono due panel, uno per ospitare la toolbar con i componenti da inserire, l'altro sul quale posare i componenti scelti.
A questo punto si aggiungano tre pulsanti alla toolbar, il primo per indicare che nessun componente è stato selezionato, il secondo per inserire un *TButton*, l'ultimo per inserire un *TEdit*.
I tre toolbutton devono avere le proprietà *AllowAllUp* e *Grouped* settate a *true*, il tutto per ottenere un effetto simile a quello dell'IDE di Delphi. Si setti per ogni bottone della barra un tag differente partendo da uno.
Il primo pulsante avrà tag uguale a uno, il secondo uguale a due e così via. Si dichiari la variabile Selected di tipo intero e si inserisca nell'evento *Click* del primo pulsante la riga seguente:

```
Selected:=TToolButton(Sender).Tag;
```

Una volta associato l'evento ad ogni pulsante della toolbar, cliccando su un tasto, la variabile *Selected* assumerà il valore del tag del pulsante premuto. Si dichiari la variabile *ControlRef* di tipo *TControlClass*.
Nell'evento *MouseDown* del panel che ospiterà i controlli selezionati scriviamo:

```
case Selected of
  2:
    ControlRef:=TButton;
  3:
    ControlRef:=TEdit;
end;
```

Mediante questo codice, a seconda del valore assunto da *Selected*, la variabile *ControlRef* assumerà un valore differente, il quale coincide con il *ClassType* del componente che si vuole inserire. Prima di proseguire si dichiarino due variabili, Counter e *ControlName*, rispettivamente di tipo intero e di tipo stringa.
Si aggiunga il codice seguente all'evento *MouseDown:*

```
if Selected > 1 then
  with ControlRef.Create(Self) do
    begin
      Inc(Counter);
      Visible:=false;
      Parent:=Self;
      ControlName:=ControlRef.ClassName;
      Delete(ControlName,1,1);
      Name:=ControlName + IntToStr(Counter);
      Left:=X;
      Top:=Y;
      Visible:=true;
      tbNone.Down:=true;  //tbNone = name del primo toolbutton
      tbNoneClick(tbNone);
    end;
```

Tramite l'utilizzo della Class Reference, si può notare come si possa aggiungere qualsiasi controllo usufruendo dello stesso codice.

ASP

SEGUIAMO LE NOSTRE APPLICAZIONI ASP PASSO PASSO

L'utility proposta è un sistema di debug delle applicazioni internet, anche una volta installate su un server remoto. La soluzione prevede la creazione di una pagina in cui mostrare il contenuto di un'area di sessione che viene valorizzata da tutte le pagine in cui vogliamo tracciare l'attività dell'applicativo, mediante una semplice chiamata alla subroutine: debug(mioValore); es: istruzioni sql e tempo d' esecuzione,variabili,errori,parametri ricevuti etc.. L'unica avvertenza da rispettare è quella di aprire la pagina di debug (*show_debug.asp*) con *Ctrl-N* o al menu : *File->Nuova Finestra* in modo da avere due istanze dell'Internet Explorer che condividono la medesima sessione.

Tip fornito dal sig. L.Civerra

1) Definiamo del global.asa due variabili di sessione:

```
Sub Session_OnStart
 .........
 Session("cntItemDebug")=0
 Session("debugArea")=""
End Sub
```

2) Creiamo un modulo da includere in tutte le pagine in cui vorremmo utilizzare il debug in linea

```
(<!--#include file="myDebug.asp"-->)
`myDebug.asp
sub debug(aMsg)
    On error resume next
    Session("debugArea")= Session("debugArea") & ">>"& aMsg
                                                  & "<br>"
    Session("cntItemDebug")=Session("cntItemDebug")+1
    '
    ' svuotiamo ad intervalli regolari la nostra area di sessione
    ' per evitare pericolosi sovraccarichi
    '
    if Session("cntItemDebug") > 30 Then
       Session("cntItemDebug") =0
       Session("debugArea")= ""
    End if
end sub
```

3) Creiamo una pagina in cui mostrare la nostra area di debug e "dotiamola" di un meccanismo di refresh ad intervalli regolari.

```
`show_debug.asp
```

IL MENU SCOMPARE

Questo codice consente di rimuovere i menu dal menu di sistema delle finestre.

Tip fornito dal Sig. S. Tomaselli

```
Declare Function GetSystemMenu Lib "user32" (ByVal hwnd As
                              Long, ByVal bRevert As Long) As Long
Declare Function DeleteMenu Lib "user32" (ByVal hMenu As
          Long, ByVal nPosition As Long, ByVal wFlags As Long) As Long
Declare Function SetMenuItemBitmaps Lib "user32" (ByVal hMenu As
                    Long, ByVal nPosition As Long, ByVal wFlags As Long, ByVal
   hBitmapUnchecked As Long, ByVal hBitmapChecked As Long) As Long
Declare Function GetMenuCheckMarkDimensions Lib "user32" () As Long
Public Const MF_BYCOMMAND = &H0&
Public Enum MnuItems
 MArrange = &HF110
 MCLOSE = &HF060
 MMAXIMIZE = &HF030
 MMINIMIZE = &HF020
 MSIZE = &HF000
 MMOVE = &HF010
 RESTORE = &HF120
End Enum
Public Sub RemoveMnu(hwnd As Long, MnuI As MnuItems)
hSysMenu = GetSystemMenu(hwnd, False)
bReturn = DeleteMenu(hSysMenu, MnuI, MF_BYCOMMAND)
End Sub
```

Il controllo di una matrice di LED

Un pannello pubblicitario comandato dal PC

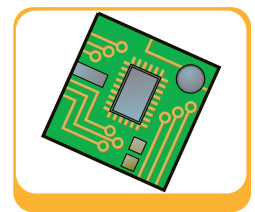
Chi vuole scoprire come realizzare un sistema modulare per realizzare un display pubblicitario a matrice di LED, legga queste pagine ed avrà modo di apprenderne le modalità costruttive.

La comunicazione riveste un ruolo fondamentale, nella società contemporanea e in tutti i settori delle attività umane. L'informazione che si desidera trasmettere risulta recepibile, con maggiore efficacia, se avviene per mezzo del maggior numero possibile di 'canali' di comunicazione. Il messaggio ha maggiore efficacia se viene trasmesso non soltanto attraverso testo scritto, ma anche per mezzo di immagini, audio e forse in futuro da messaggi olfattivi e tattili. Pensiamo al realismo che avrebbe un film visto al cinema in tre dimensioni con la possibilità di percepire anche gli odori, la temperatura e le sensazioni tattili di una particolare scena. L'impatto visivo senz'altro fornisce un flusso di informazioni più elevato di qualunque altro canale di comunicazione. In effetti il detto inglese *'One picture is worth one thousand words'* rende perfettamente l'idea di quello che intendiamo dire. Per questo motivo siamo ormai circondati, durante la vita di tutti i giorni da schermi, display digitali e pannelli pubblicitari di ogni genere. In questa sede vogliamo progettare un sistema modulare per la realizzazione di un pannello pubblicitario a matrice di LED controllato da un Personal Computer attraverso la porta parallela. Analizzeremo le metodologie di implementazione hardware, studiando il circuito elettronico dell'interfaccia Display-PC e sviluppando il relativo software di controllo. Lo studio del problema si completerà con la realizzazione pratica del progetto, essendo dell'opinione che soltanto l'applicazione pratica porta alla conoscenza profonda di un determinato argomento, come ben espresso dalla nota massima di Confucio : *"Ascolto e dimentico, Vedo e ricordo, Faccio e capisco"*.

Fig. 1: **Per maggiori informazioni sul Pannello a matrice di LED e sull'apparecchiatura 'PC Explorer light' è possibile visitare il sito: www.pcexplorer.it.**

Fig. 2: **La matrice di LED mostrata in figura è modulare, per ottenere un display di grandi dimensioni è sufficiente collegare più moduli in serie. Gli anodi delle colonne sono connessi insieme, così come i catodi.**

LA FILOSOFIA DI IMPLEMENTAZIONE

La realizzazione di un pannello pubblicitario a matrice di LED può essere realizzato in moltissimi modi, utilizzando una notevole quantità di differenti tecniche di implementazione e con un notevole assortimento di differenti componenti elettronici.
Esistono, senz'altro, circuiti integrati adatti all'interfacciamento diretto di un display come quello che intendiamo realizzare, sono disponibili inoltre pannelli già pronti all'uso. Dal nostro punto di vista, desideriamo progettare una applicazione che sia concretamente realizzabile dal lettore, per mezzo di componenti elettronici sicuramente e facilmente reperibili, facilitando e rendendo la filosofia di costruzione il più possibile semplice e diretta.
In questa sede vogliamo semplificare, al massimo, il

L'autore è lieto di rispondere ai quesiti dei lettori sull'interfacciamento dei PC all'indirizzo: luca.spuntoni@ioprogrammo.it

I COMPONENTI NECESSARI:

N 1 CMOS 4017
N 50 Diodi LED Rossi
N5 Res. 470 Ohm

I componenti sono reperibili presso il sito www.rs-components.it

progetto del circuito utilizzando un solo circuito integrato CMOS 4017, dotato di un contatore di Johnson a cinque stadi, completi di decodificatore a dieci uscite, che utilizziamo per pilotare il nostro modulo per pannelli pubblicitari. Per rendere quanto abbiamo detto sicuramente verificabile dal lettore, diciamo che i pochi componenti elettronici che utilizzeremo sono reperibili in qualunque negozio di componenti elettronici oppure per corrispondenza presso la RS-Components (*www.rs-components.it*); l'assemblaggio del circuito può essere effettuato senza saldature per mezzo dell'apparecchiatura PCExplorer light reperibile sul sito *www.pcexplorer.it*.

N Pin Modulo LED	Descrizione	N Pin Modulo LED	Descrizione
1	Catodo Riga N 0	9	Anodo Colonna 6
2	Catodo Riga N 1	10	Anodo Colonna 5
3	Catodo Riga N 2	11	Anodo Colonna 4
4	Catodo Riga N 3	12	Anodo Colonna 3
5	Catodo Riga N 4	13	Anodo Colonna 2
6	Anodo Colonna 9	14	Anodo Colonna 1
7	Anodo Colonna 8	15	Anodo Colonna 0
8	Anodo Colonna 7	16	Non collegato

Tab. 1: Connessioni del modulo a matrice di LED.

I MODULI A MATRICE DI LED

È stato accennato al fatto che il sistema è modulare: vediamo che cosa intendiamo esattamente con quanto abbiamo appena detto. Il display che intendiamo realizzare potrebbe variare per dimensioni anche in maniera considerevole: per qualche lettore potrebbe essere sufficiente realizzare un piccolo display multi-funzione per visualizzare lo stato operativo di una macchina utensile, mentre altri potrebbero desiderare la realizzazione di un vero e proprio schermo video a matrice di LED, con capacità di visualizzare immagini televisive. Il singolo modulo può essere facilmente realizzato dal lettore per mezzo di una piastra millefori, posizionandovi una matrice di 5 x 10 LED rossi e collegando gli anodi di ogni singola colonna insieme, analogamente ai catodi di ciascuna riga: in alternativa è possibile richiedere la disponibilità del kit completo inviando una e-mail all'indirizzo *luca.spuntoni@ioprogrammo.it.* Per comprendere meglio il funzionamento di questi moduli analizziamone la struttura osservando la *Tabella 1*. Supponiamo di volere accendere il LED collocato nell'ottava colonna e nella seconda riga: innanzi tutto notiamo che la numerazione delle righe e colonne inizia con "0", quindi dobbiamo considerare le coordinate 7,1 ai fini dell'individuazione del LED da accendere. Per comandare l'accensione del LED 7,1 è sufficiente collegare l'anodo della colonna N 7 (Pin 8) a +5 V attraverso una resistenza da 470 ohm ed il catodo della riga N 1 (Pin 2) a massa come mostrato in Fig. 3. Per comandare l'accensione dei 50 led in modo indipendente con questo metodo, che abbiamo utilizzato per comprendere il funzionamento del modulo avremmo bisogno di 50 resistenze di carico e di una complessa elettronica di commutazione, realizzabile, ma complessa e costosa. Per semplificare l'elettronica di controllo utilizziamo un metodo di scansione sequenziale commutando una colonna per volta e comandando di volta in volta l'accensione dei LED delle righe interessate: in questo modo possiamo utilizzare soltanto 5 (cinque) resistenze di carico, risparmiandone ben 45 per ogni modulo utilizzato. Per fare questo utilizziamo un circuito integrato molto diffuso che funge da contatore 'decadico' come meglio descriviamo di seguito.

NOTA

ACQUISTARE PC EXPLORER LIGHT

L'apparecchiatura PC Explorer light è prodotta e commercializzata dalla Elisys s.r.l. e può essere acquistata sul web all'indirizzo www.pcexplorer.it inviando una e-mail all'indirizzo pcexplorer@elisys.it, oppure telefonicamente al numero 0823/468565 o via Fax al: 0823/495483.

Fig. 3: *Per comandare l'accensione di un unico LED della matrice, ad esempio il dot 8,2 è sufficiente effettuare le connessioni riportate in figura.*

IL CIRCUITO INTEGRATO 4017

Il circuito integrato 4017 contiene, al suo interno, un contatore di Johnson a cinque stadi, munito di un decodificatore decimale a dieci uscite O0- O9 attive a livello logico HIGH: comprende inoltre una linea di riporto logico chiamata O5-9, che però non viene utilizzata nell'applicazione che proponiamo in questa sede. L'avanzamento del conteggio avviene quando sul piedino *CP0* avviene una transizione dello stato logico *LOW->HIGH*, mentre *CP1/* è a livello logico LOW, oppure quando su *CP1/* avviene una transizione *HIGH->LOW* mentre su *CP0* si ha uno stato logico *HIGH*. Sul terminale *O5-9* si ha uno stato logico *LOW,* quando sono attive le linee *O0 –O4*, mentre si ha la presenza di un livello *HIGH* quando sono attive le linee *O5 – O9:* questo contatto del circuito integrato permette di espandere lo schema elettrico rendendo possibile l'aggiunta di eventuali sezioni in cascata, che permettono di incrementare il numero di luci controllate. La tabella della verità del circuito integrato mostra le funzioni logiche del dispositivo, in particolare notiamo che, quando il piedino MR (*Master Reset*) viene posto a livello logico HIGH, indipendentemente dallo stato delle altre linee, la logica interna del chip si pone nello stato iniziale del conteggio, ovvero con *O0* e *O5-9* a livello *HIGH* e con le altre uscite a livello *LOW.*

ANALISI DELLO SCHEMA ELETTRICO

Analizzando lo schema elettrico della nostra realizzazione ne apprezziamo immediatamente la semplicità ed il numero ridotto di componenti che si limita a 50 LED, 5 resistenze ed un solo circuito integrato. In alto si notano le connessioni con la porta parallela del PC attraverso l'apparecchiatura PCExplorer light. Il Pin *D3* rappresentante il bit corrispondente della porta dati nello standard Centronics, viene utilizzato come CLOCK per il contatore decadico, mentre il Pin *D5* funge da 'Master Reset' del componente, azzerando il conteggio quando viene attivato. Sul lato sinistro dello schema i Pin *D0-D2* e *D6-D7* rappresentano i segnali di controllo dell'accensione delle righe 0-4 del modulo LED, in funzione della colonna attiva in un dato momento stabilita dal conteggio dell'integrato 4017. Lo schema elettrico è stato semplificato al massimo, utilizzando l'integrato 4017 per comandare direttamente l'accensione dei LED, per aumentare la luminosità del pannello occorre inserire per ogni colonna un semplice circuito di commutazione a transistor. La modularità del sistema si realizza collegando più moduli in serie, sfruttando il pin di riporto dell'integrato 4017 (*/O5-9* Pin 12) che deve essere collegato all'ingresso dei moduli successivi attraverso una opportuna logica di controllo.

REALIZZAZIONE DEL CIRCUITO ELETTRONICO

Il circuito può essere realizzato seguendo le connessioni riportate nello schema elettrico e le immagini riportate in queste pagine, che per comodità del lettore sono state incluse nel CD-Rom allegato alla rivista, nonché sul Sito Web *www.ioprogrammo.it*, con il nome *'Immagini_Pannello_Elettronico.zip'*. Nello schema il circuito integrato è stato raffigurato come nella realtà, ovvero la piedinatura è stata posizionata come sul chip vero e proprio, inoltre anche la disposizione dei componenti è stata rappresentata in linea di massima come sulla piastra sperimentale, per facilitare al massimo la realizzazione del circuito. La lista dei componenti necessari viene riportata a lato di queste pagine e nel circuito elettrico, per comodità e su richiesta dei lettori all'interno del file: *'SpuntoPannelloPubblicitario.zip'*, con il nome: *'Schema_Elettrico_Pannello_Pubblicitario.bmp'*. Sul lato sinistro dello schema si possono notare le connessioni alle linee relative alla porta parallela e di alimentazione dell'apparecchiatura '*PC Explorer light*', sulla quale è possibile avere maggiori informazioni visitando il sito: *www.pcexplorer.it*. La realizzazione può avvenire, facilmente, posizionando, innanzi tutto, il circuito integrato, e le cinque resistenze, effettuando tutte le connessioni elettriche prima di connettere il pannello a matrice di LED. Il cablaggio è stato realizzato utilizzando l'apparecchiatura PCExplorer light, in alternativa è possibile utilizzare le tecniche costruttive convenzionali, ovvero dotandosi di stagno, saldatore ed una buona dose di pazienza: il lettore ha in ogni caso tutte le informazioni necessarie alla realizzazione della parte hardware e più avanti troverà il software di gestione, completo di componenti pronti all'uso, del programma compilato e funzionante dotato di codice sorgente.

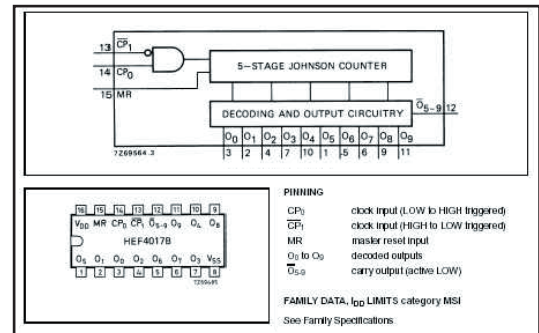
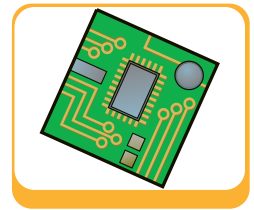



Fig. 4: Lo schema logico e la piedinatura del circuito integrato HEF4017, reperibili in forma completa su Internet all'indirizzo: www.components.philips.com/ (cortesia Philips Semiconductors).

IL PROGRAMMA C++ DI CONTROLLO DEL CIRCUITO

Il codice sorgente, scritto in C++ viene messo a completa disposizione del lettore: *SpuntoPannelloPubblicitario.zip*: in questa sede analizziamo soltanto le parti più significative, rimanendo a disposizione del lettore per ogni chiarimento all'indirizzo: *luca.spuntoni@ioprogrammo.it*. Il software di controllo è stato collaudato con Win 3.x, Win 9x e Win Me, se si utilizza Win 2000, XP oppure NT, è possibile utilizzare un driver, per evitare l'errore di *'Privileged Instruction'* generato da questi ultimi sistemi operativi quando si tenta di accedere alle porte hardware del PC quale *'PortTalk'* (*PortTalk22.zip*), scaricabile del sito: *www.beyondlogic.org*. Nella parte iniziale del programma si notano i componenti *'SpuntoLedComponent'* e *'TSpuntoHardwarePortIO_unit'* che contengono i componenti di gestione della simulazione software dei LED presenti nel programma e la gestione delle porte hardware.

```
//--------------------------------------------------------//
//      Spuntosoft LED SCREEN Controller
//      Version 1.0  November 2003
//      Luca Spuntoni all rights reserved
#include <vcl.h>
#pragma hdrstop
#include "SpuntoPannelloPubblicitarioUnit.h"
#pragma package(smart_init)
#pragma link "SpuntoLedComponent"
#pragma link "TSpuntoHardwarePortIO_unit"
```

FUNCTION TABLE

MR	CP_0	$\overline{CP_1}$	OPERATION
H	X	X	$O_0 = \overline{O_{5\text{-}9}} = H$; O_1 to $O_9 = L$
L	H	↓	Counter advances
L	↑	L	Counter advances
L	L	X	No change
L	X	H	No change
L	H	↑	No change
L	↓	L	No change

Notes
1. H = HIGH state (the more positive voltage)
2. L = LOW state (the less positive voltage)
3. X = state is immaterial
4. ↑ = positive-going transition
5. ↓ = negative-going transition

Fig. 5: Nell'immagine di figura si riporta la tabella della verità dell'integrato HEF4017 (cortesia Philips Semiconductors).

```
#pragma resource "*.dfm"
unsigned short Datain, ColumnNumber;
unsigned short LPTbaseAddress, LPT1DataAddress,
                    LPT1StatusAddress, LPT1ControlAddress;
unsigned short  ArrayDataOut[5][10]={
                      {0,0,0,0,0,0,1,0,0,0},  // Screen Array
              {0,1,1,1,1,1,1,1,0,0}, {0,1,1,1,1,1,1,1,1,0},
               {0,1,1,1,1,1,1,1,0,0}, {0,0,0,0,0,0,1,0,0,0}
};
```

Notiamo, inoltre, l'inizializzazione della matrice *'ArrayDataOut'* deputata a contenere l'immagine che dovrà essere inviata al pannello LED che, nell'esempio, rappresenta una freccia rivolta verso destra.
La matrice viene inizializzata direttamente per chiarezza e semplicità di trattazione, il lettore potrà sviluppare le proprie procedure di definizione dell'array in funzione della applicazione che intenderà sviluppare. All'esecuzione del programma viene lanciata la procedura *'FormCreate'* che provvede all'inizializzazione delle variabili della applicazione ed, in particolare, dei valori di default utilizzati per accedere alla porta parallela.

Fig. 8: *Lo schema elettrico del circuito di controllo che, per comodità e su richiesta dei lettori, è stato inserito all'interno del file: 'SpuntoPannelloPubblicitario.zip', con il nome: 'Schema_Elettrico_Pannello_Pubblicitario.bmp'*

```
void __fastcall TSpuntoLEDSCREENForm::FormCreate(
                                          TObject *Sender)
{ // Sets the labels
Label1->Caption = IntToStr(PeriodTrackBar->Position);
Label9->Caption = IntToStr(PulseWidthTrackbar->Position);
// Sets the Power LED
if(PowerONSpeedButton->Down)
  SpuntoPowerLed->LedOn();
else
  SpuntoPowerLed->LedOff();
// Default (LPT1) Port setup
  LPTbaseAddress=0x0378;
  LPT1DataAddress=LPTbaseAddress;
  LPT1StatusAddress=LPTbaseAddress+1;
  LPT1ControlAddress=LPTbaseAddress+2;
//Array Management
  ColumnNumber=0; }
//-----------------------------------------------------------
```

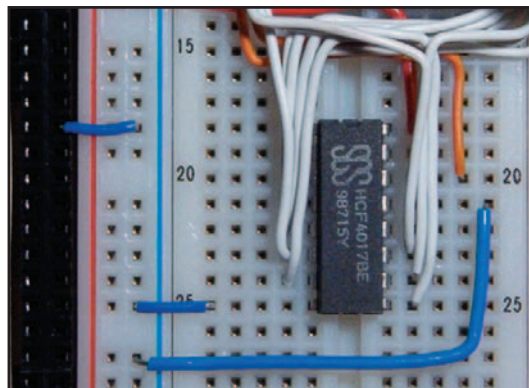



Fig. 7: *I collegamenti nell'intorno del circuito integrato sono molto densi: per maggiori dettagli fare riferimento allo schema elettrico.*

NOTA

IL MODULO PER PANNELLI PUBBLICITARI IN AZIONE

Nel CD allegato alla rivista e su www.ioprogrammo.it è disponibile un filmato che mostra il pannello a matrice di LED in funzione (*Modulo_Display_LED.AVI*).

Il cuore del programma è la procedura *'MainTimerTimer'* che gestisce l'invio del CLOCK di conteggio all'integrato 4017 attraverso la gestione dell'evento di timing generato da *'MainTimer'*, nonché provvede a leggere la matrice *'ArrayDataOut'* ed ad attivare la sequenza opportuna per l'accensione della matrice LED. Si noti che, per attivare l'accensione di una riga, ad esempio la riga '0' è sufficiente operare l'*OR* logico tra il valore presente sulla porta ed il valore binario *'00000001'*. Analogamente per spegnere la stessa riga basta effettuare l'*AND* logico tra il valore presente sulla porta ed il valore binario *'11111110'*. Questo concetto viene applicato a tutte le righe in sequenza: non è stato utilizzato un ciclo *For*, dal momento che l'implementazione hardware ci costringe, per mantenere la compatibilità con altre schede presentate in articoli precedenti, ad utilizzare una serie di bit non sequenziale. Il codice potrebbe essere notevolmente ottimizzato, ma si è preferito fornire una maggiore chiarezza e leggibilità del codice.

```
void __fastcall TSpuntoLEDSCREENForm:
                        :MainTimerTimer(TObject *Sender)
{  //Main Timer
  if (PowerONSpeedButton->Down)  //Power is ON
 {  //*** DATA ***
    DelayTimer->Enabled=true;
    Label15->Caption = IntToStr(ColumnNumber);
 ColumnNumber=ColumnNumber + 1;
 Datain=SpuntoHardwarePort->ReadPort(
                                         LPT1DataAddress);
 if (ArrayDataOut[0][ColumnNumber]==1)    // Row 0
 { Datain=(Datain & 0xFE);      //  11111110 }
 else
 {
 Datain=(Datain | 0x01);       // 00000001 }
 if (ArrayDataOut[1][ColumnNumber]==1)   // Row 1
 { Datain=(Datain & 0xFD);      //  11111101 }
 else
 { Datain=(Datain | 0x02);       // 00000010 }
 if (ArrayDataOut[2][ColumnNumber]==1)    // Row 2
 { Datain=(Datain & 0xFB);      //  11111011 }
 else
 { Datain=(Datain | 0x04);       // 00000100 }
 if (ArrayDataOut[3][ColumnNumber]==1)    // Row 3
 { Datain=(Datain & 0xBF);      //  10111111 }
 else
 { Datain=(Datain | 0x40);       // 01000000 }
 if (ArrayDataOut[4][ColumnNumber]==1)    // Row 4
 { Datain=(Datain & 0x7F);      //  01111111 }
 else
 { Datain=(Datain | 0x80);       // 10000000 }
 //*** CLOCK ***
 SpuntoHardwarePort->LedOn();
 Datain=(Datain | 0x08);      // Clock bit to 1
 SpuntoHardwarePort->WritePort(LPT1DataAddress,Datain);
 if (ColumnNumber==10)              // Last Row
 {  //*** Reset ***
```

```
 StrobeSpuntoLed->LedOn();
 Datain=SpuntoHardwarePort->ReadPort(LPT1DataAddress);
 Datain=(Datain | 0x20); //  Resets the 4017 counter
                                         (Reset is 1)
 SpuntoHardwarePort->WritePort(LPT1DataAddress,Datain);
 // Bit counter reset
 ColumnNumber=0; // Resets the bit position }
}
 else
 { SpuntoHardwarePort->LedOff(); // Power is OFF
   DelayTimer->Enabled=false; }
}
```

Al termine del conteggio dell'integrato 4017 , ovvero al raggiungimento della decima colonna si opera il RESET del contatore riportando la scansione alla colonna '0'.

```
//-----------------------------------------------------------
void __fastcall TSpuntoLEDSCREENForm::DelayTimerTimer(
                                         TObject *Sender)
{ //Delay timer
 Datain=SpuntoHardwarePort->ReadPort(LPT1DataAddress);
 Datain=(Datain & 0xC7);        // Clock is 0
 SpuntoHardwarePort->WritePort(LPT1DataAddress,Datain);
 SpuntoHardwarePort->LedOff();
 StrobeSpuntoLed->LedOff();
 DelayTimer->Enabled=false;
}
//-----------------------------------------------------------
```

La procedura *DelayTimerTimer* viene lanciata allo scatenarsi dell'evento generato da *DelayTimer*, che ha lo scopo di realizzare la parte negativa delle forme d'onda che vengono inviate al circuito di conteggio ed in particolare del segnale di *CLOCK*.

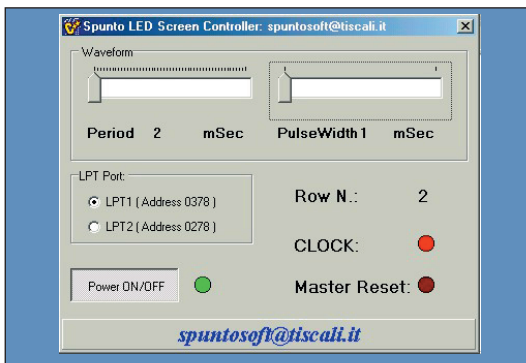



Fig. 8: ***Nella figura si ha una schermata del programma che controlla il circuito: è possibile controllare la velocità di esecuzione della sequenza delle luci per mezzo dei cursori, nonché scegliere la porta parallela con la quale controllare l'apparato.***

COLLAUDO DEL SISTEMA

Una volta completata la nostra realizzazione, siamo giunti al momento di collaudarne il funzionamento. Provvediamo a verificare un'ultima volta tutte le connessioni elettriche, completato il controllo, siamo pronti a collegare il circuito alla porta parallela del PC, ovviamente a computer rigorosamente spento. Accendiamo il calcolatore e lanciamo il programma di controllo, nonché quello di monitor della porta seriale, contenuti nel file '*SpuntoPannelloPubblicitario.zip*' e provvediamo subito a selezionare la porta parallela sulla quale è collegato il circuito da verificare. Premiamo ora il pulsante '*ON/OFF*' ed alimentiamo il circuito elettronico. Se tutto funziona come deve, dovremmo vedere accendersi sul pannello LED una sequenza tale da visualizzare una freccia rivolta verso destra. Lo schema può essere ampliato notevolmente, fino alla realizzazione di pannelli di grandi dimensioni.

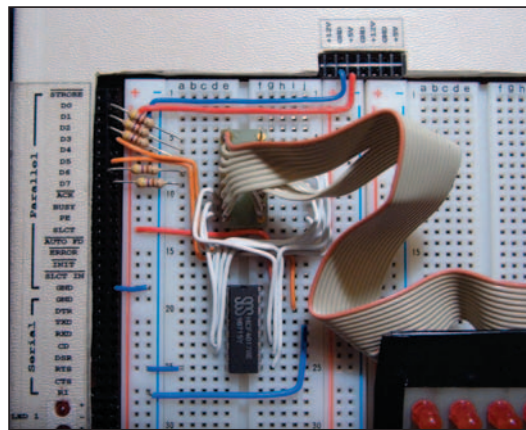
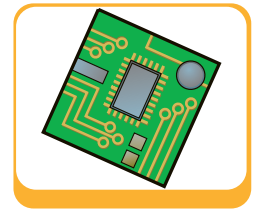



Fig. 8: ***Collegando il connettore del modulo a matrice di LED siamo pronti ad utilizzare l'applicazione.***

CONCLUSIONI

Nonostante i limiti di spazio di trattazione abbiamo analizzato in questa sede come realizzare un sistema modulare per schermi a matrice di LED. Il progetto dello schema elettrico, tutti i collegamenti necessari, il software compilato ed i relativi codici sorgenti sono stati messi a completa disposizione del lettore: due filmati dimostrativi sono disponibili sul CD ROM allegato alla rivista. Il lettore vorrà comprendere che, nonostante quanto esposto in queste pagine sia stato debitamente verificato e collaudato, tuttavia viene riportato a scopo illustrativo e di studio, pertanto l'editore e l'autore non sono da considerare responsabili per eventuali conseguenze derivanti dell'utilizzo di quanto esposto in questa sede, soprattutto per la tipologia e la complessità dell'argomento.
L'autore è lieto di rispondere ad ogni richiesta di chiarimento o delucidazione sull'argomento all'indirizzo di posta elettronica *luca.spuntoni@ioprogrammo.it*.

Luca Spuntoni

• CONTROLLIAMO LA PORTA PARALLELA CON DELPHI 6
Luca Spuntoni
ioProgrammo N57-58 Aprile e Maggio 2002

PRECAUZIONI

Prima di collegare il circuito al nostro PC occorre verificare la nostra realizzazione con attenzione per assicurarci che tutto sia stato collegato come previsto. L'utilizzo del programma presentato in questa sede mentre è collegata una qualunque altra periferica al PC sulla porta LPT1, può bloccarne il funzionamento.

Interazione client server

PocketPC e DB

Tecniche di ricezione e trasmissione di file multimediali con dispositivi PocketPC, su base client/server: sviluppiamo un'applicazione SQL Server.

HARDWARE:
Pocket PC o (dispositivo Windows CE Based), Computer con almeno processore Pentium II e 128 MB di memoria.

SOFTWARE:
Windows 98 SE /2000/XP, IIS (oppure PWS-Personal WEB Server con Windows 9x), SQL Server 2000 con Service Pack 1 o superiore, SQL Server CE 2.0, Embedded Visual Tools.

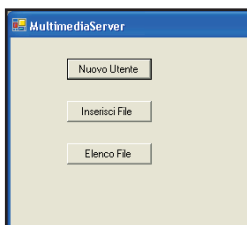



Fig. 2: *Maschera iniziale della applicazione.*

La parte Server dell'applicazione sarà il cuore del sistema che permetterà ai client PocketPC di "vedere" le risorse multimediali e di selezionarle per un downloading sul dispositivo. La tecnologia SQL Server consentirà di costruire una applicazione veramente professionale. Nel numero 74 (novembre), abbiamo costruito l'applicazione *MultimediaClient* per PocketPC 2002 costruendo la classe *CMediaOperation*. Quest'ultima classe consentiva non solo di memorizzare un media file sul palmare nel *Database* locale, ma anche di leggere un qualsiasi file per visualizzarne o ascoltarne il contenuto. In questo appuntamento, completeremo l'applicazione *MultimediaServer*. Nella fase di implementazione dell'applicazione si renderà necessario effettuare opportune operazioni su SQL Server allo scopo di far funzionare l'intero meccanismo di trasmissione dei dati. Svilupperemo la parte Server della Mobile application in tecnologia .NET, usando il linguaggio C#. Le funzionalità che andremo a costruire sul Server sono le seguenti:

- creazione degli utenti abilitati al servizio di condivisione risorse;
- aggiunta dei file da rendere disponibili per il download dei dispositivi mobili;
- visualizzazione dell'elenco dei file condivisibili con i dispositivi Pocket PC.

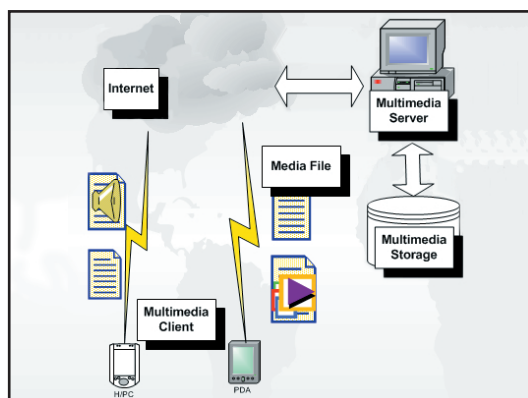
Per quanto riguarda SQL Server 2000 vedremo come aggiungere filtri dinamici alla pubblicazione di un Database. È necessario evidenziare che i filtri dinamici consentiranno di selezionare in maniera opportuna i dati da inviare ai diversi client.

Fig. 1: *L'architettura della intera applicazione.*

APPLICAZIONE MULTIMEDIASERVER

Iniziamo con la creazione della applicazione Server *MultimediaServer.* Per quanto riguarda lo schema del database *MultimediaStorage,* esso può essere ricostruito utilizzando il relativo file di script (*MultimediaStorage.sql*) nel CD allegato. Lo script va lanciato utilizzando la utility Query Analyze di SQL Server. Abbiamo deciso di sviluppare l'applicazione su piattaforma .NET. A tal fine apriamo un nuovo progetto in Visual Studio .NET di tipo Windows Application nel linguaggio C#. Nella maschera iniziale, possiamo notare le tre funzionalità base che andremo a sviluppare nel corso della nostra analisi. La prima funzionalità che prendiamo in considerazione è il censimento di un nuovo palmare. La funzionalità in questione è molto importante poichè ci permetterà di comprendere come implementare meccanismi di sicurezza in SQL Server 2000. In particolare, nel metodo di salvataggio dei dati del palmare e del relativo utilizzatore associato sarà richiamata una apposita stored procedure sul Server di Database che conterrà non solo la logica di inserimento di questi dati ma anche il codice necessario per la creazione di un nuovo utente di SQL Server univocamente associato al palmare. Il requisito di creare un nuovo utente si SQL Server per ogni palmare che si vuole acceda al servizio, migliora la sicurezza nell'accesso ai dati. Infatti, nel caso si volesse disabilitare un palmare all'accesso ai dati multimediali, sarà sufficiente individuare il relativo utente di database ed disabilitarlo nelle sue funzioni.

REGISTRAZIONE NUOVO UTENTE

Vediamo in dettaglio la parte della applicazione necessaria per la registrazione di un nuovo palmare e

del relativo utente. Il form "Nuovo Utente" (oggetto *frmNuovoUtente* nel Workspace dell'applicazione *MultimediaServer*) consente di inserire le informazioni relative al proprietario del palmare: *Nome, Cognome, password* e *userid*. I dati di userid e password ci potrebbero servire per effettuare il Login. Il form di Login e la relativa logica vengono lasciati al lettore come facile estensione dell'applicazione, non volendo sottrarre spazio alla implementazione delle altre funzionalità. Dal form "Nuovo Utente" possiamo notare la presenza dei tasti *Annulla, Salva* e *Associa Palmare*. Il pulsante *Salva* consente il salvataggio delle informazioni dell'utente. Mentre il pulsante *Associa Palmare* permette di associare logicamente un palmare all'utente stesso. L'operazione di associazione di un palmare ad un utente è possibile solo dopo l'inserimento dell'utente, se così non fosse l'utente verrebbe avvertito da un messaggio di errore. Abbiamo modellato il fatto che sia avvenuto l'inserimento dei dati del nuovo utente, dichiarando nella classe *frmNuovoUtente* il tipo enumerato seguente :

```
public enum STATI{SALVATO=0, NON_SALVATO=1};
```

nella variabile di classe *m_statoForm* abbiamo memorizzato lo stato corrente della form. Inizialmente abbiamo posto la variabile nello stato *NON_SALVATO*. Il form passa nello stato *SALVATO* non appena l'operazione di inserimento dei dati del nuovo utente avviene con successo. È bene precisare che, in ogni form del workspace dell'applicazione, abbiamo posto una variabile di classe *m_connesione* di tipo *SqlConnection* che rappresenta la connessione attiva al Database *MultimediaStorage*. Per semplificare la procedura di connessione al database abbiamo costruito la classe *DBSetting* il cui schema è in Fig. 4. Dalla stessa figura possiamo notare le properties statiche della classe:

1. **Database:** rappresenta il nome del Database che contiene i dati (nel nostro caso *MultimediaStorage*).
2. **Provider:** rappresenta il nome della macchina che assume il ruolo di Server di Database con SQL Server 2000;
3. **UserId:** User Id dell'utente abilitato all'accesso in SQL Server;
4. **Password:** relativa password dell'utente abilitato;
5. **StringaDiConnessione:** il parametro da passare al costruttore dell'oggetto *m_connessione* in ogni form.

Le prime quattro properties della classe *DBSetting* vengono settate solo all'inizio della applicazione nel costruttore della classe relativa al form iniziale (classe *frmPrincipale*). A questo punto è possibile prelevare la stringa di connessione e istanziare l'oggetto connessione:

```
m_connessione = new SqlConnection(
                                DBSetting.StringaDiConnessione);
```

Possiamo notare come la property sia ottenuta direttamente sul nome della classe *DBSetting* dato che si tratta di una property statica. Il metodo di salvataggio dei dati dell'utente utilizza la stored procedure *spNuovoUtente* che riceve in ingresso i dati inseriti nella form e restituisce in caso di successo il codice dell'utente appena memorizzato nella tabella *UTENTI* del database e una stringa contente il messaggio di errore in caso di fallimento. Il codice necessario per l'invocazione della stored procedure *spNuovoUtente* e lagica necessaria per l'analisi dei dati restituiti possono essere trovati nel gestore dell'evento *onClick* del pulsante *Salva* della classe *frmNuovoUtente* : metodo *btnSalva_Click*. Diamo solo un breve commento della logica implementata nel metodo: prima di tutto viene verificato che il form non si trovi nello stato *SALVATO*, se così fosse si esce dal metodo altrimenti si inserirebbe due volte uno stesso utente. A questo punto inizia la logica di inserimento richiamando la stored procedure con l'oggetto *cmdInsUtente* di tipo *SqlCommand*. Dopo la fase di passaggio dei parametri viene aperta la connessione al database e mandata in esecuzione la stored procedure. Se tutto è andato a buon fine la linea di codice

```
m_codUtente = (int)
    cmdInsUtente.Parameters["@p_codUtente"].Value;
```

permette di recuperare il codice dell'utente appena inserito e memorizzarlo nella variabile di classe *m_codUtente* di *frmNuovoUtente*. Quest'ultima informazione sarà poi necessaria per l'inseriemento del palmare associato all'utente dato il vicolo di chiave esterna esistente tra le tabelle *UTENTI* e *PALMARI* di *MultimediaStorage*.

CREAZIONE DI UN UTENTE IN SQL SERVER

In questa sezione ci occupiamo in dettaglio delle operazioni da effettuare per la creazione di un utente di database da associare ai dati di un nuovo palmare inseriti nella form "Nuovo Palmare" (Fig 3b). Questa form è attivata dopo il click sul tasto della form "Nuovo Utente". Per censire un palmare occorre specificare oltre una sua descrizione anche le informazioni di *DB User ID* e *DB Password* che rappresentano i dati relativi al nuovo utente di SQL Server che andremo a creare. Le credenziali del nuovo utente saranno utilizzate per effettuare l'accesso du-

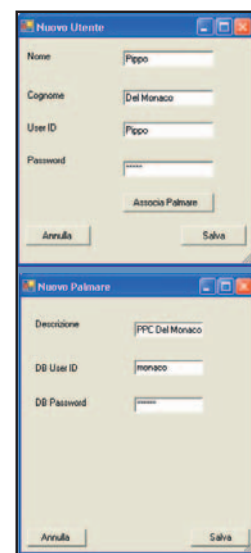



Fig. 3: Le due maschere relative al censimento di un nuovo palmare.

Utilizzare la pubblicazione dei dati di un database insieme al meccanismo dei filtri dinamici rende molto potenti e professionali le nostre applicazioni che devono distribuire dati tra diversi client. Nel caso dei PocketPC un siffatto meccanismo si sposa in maniera perfetta evitando di dover inventare strani artifici che appesantirebbero la manutenibilità della applicazione.

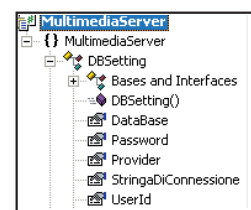



Fig. 4: Schema classe DBSetting.

SUL WEB

I filtri dinamici permettono di creare delle pubblicazioni con Merge Replication e consentono di filtrare i dati delle tabelle pubblicate in funzione del particolare sottoscrittore che accede al Database. I filtri dinamici sono "filtri di riga" e sono applicati su una singola tabella e non su join di tabelle. I benefici nell'utilizzo dei filtri dinamici sono molteplici:

1) la partizione dei dati filtrati in funzione dell'utente corrente che si collega al database, permette di utilizzare e gestire una sola pubblicazione dei dati;

2) i sottoscrittori della pubblicazione ricevono solo le informazioni di cui necessitano poichè i dati sono filtrati in virtù delle proprietà di connessione della sottoscrizione del Merge Agent. In particolare, i dati vengono valutati nel momento in cui parte il processo di Merge che sta replicando i dati tra il client sottoscrittore e il Publisher.

Users 7 Items

Name	Login Name	Database
dbo		Permit
etavolaro	etavolaro	Permit
gino	gino	Permit
jack	jack	Permit
pino	pino	Permit
pippo	pippo	Permit
topolino	topolino	Permit

Fig. 5: *Utenti di SQL Server inseriti*

rante la sincronizzazione da del client PocketPC sottoscrittore della pubblicazione del database.
Una volta inserite le informazioni precedenti basta cliccare il tasto *Salva* della form per l'attivazione del gestore di evento *btnSalva_onClick* che troviamo nella classe *frnNuovoPalmare* della *Solution* del progetto *MultimediaServer.* Il codice del gestore di evento richiamerà la stored procedure *spNuovoPalmare* di SQL Server. Vediamo in dettaglio il codice della Stored procedure:

```
CREATE PROCEDURE dbo.spNuovoPalmare
@p_codUtenteFk      int,
@p_descrizione      nvarchar(255),
@p_dbUserId         nvarchar(50),
@p_dbPsw            nvarchar(50),
@p_stringaErrore    nvarchar(255)        output,
@p_codPalmare       int                  output
 AS
Begin Tran
INSERT INTO PALMARI(DESCRIZIONE, DB_USER,
                    DB_PASSWORD, COD_UTENTE_FK,
                            ABILITATO, CANCELLATO)
VALUES(@p_descrizione,@p_dbUserId, @p_dbPsw,
                                @p_codUtenteFk,1,0)
if (@@error<>0)
begin
   set @p_stringaErrore='Problemi nello inserimento
                                       del palmare'
   rollback tran
   return 3
end
Select @p_codPalmare = cod_palmare from PALMARi
                         WHERE DB_USER = @p_dbUserId
Commit Tran
if not exists (select * from master.dbo.syslogins where
                             loginname = @p_dbUserId)
BEGIN
   exec sp_addlogin @p_dbUserId,@p_dbPsw,
                     'MultimediaStorage', null, null
END
/* Creazione utenti */
if not exists (select * from dbo.sysusers where name =
                                       @p_dbUserId)
BEGIN
   exec sp_grantdbaccess @p_dbUserId, @p_dbUserId
END
exec sp_grant_publication_access
              'MultimediaStoragePubl', @p_dbUserId
return 0
GO
```

Nella prima parte della procedura vengono inseriti i dati del nuovo palmare nella tabella *PALMARI* del Database. Nella seconda parte, effettuando una query sulla tabella *syslogins* del database *Master* di SQL Server, si verifica che non sia presente nessun altro utente con lo stesso user id passato in input.

A questo punto la creazione del nuovo utente di SQL Server avviene in tre passi, invocando le seguenti stored procedure di sistema:

1. **sp_add** login per la creazione di un utente al database *MultimediaStorage* con user id e password specificate;
2. **sp_grant_access** permette all'utente creato al passo precedente di avere i permessi di accesso al database corrente;
3. **sp_grant_publication** consente di aggiungere l'utente alla lista degli utenti della pubblicazione *MultimediaStoragePubl* del database.

Il passo successivo sarà la creazione della pubblicazione *MultimediaStoragePubl* del database contente i dati multimediali. In Fig. 5 possiamo notare la lista degli utenti di database che abbiamo inserito e relativi ad altrettanti palmari che possono accedere ai dati multimediali di condivisione. Se volessimo disabilitare l'accesso ai dati al palmare associato all'utente "pippo" potremmo richiamare la stored procedure di sistema *sp_revoke_publication* sul database dei file nel seguente modo:

```
exec sp_revoke_publication_access
                      'MultimediaStorage', pippo.
```

Questo rappresenta un modo molto elegante per proteggere i nostri dati di condivisione sul server.

LA PUBBLICAZIONE DEI DATI

Procediamo con la creazione della pubblicazione dei dati del database *Multimedia Storage.* Chiameremo la pubblicazione *MultimediaStoragePubl.*

1. Creiamo una nuova pubblicazione del database *MultimediaStorage* specificando *Merge Replication* come tipo di pubblicazione.
2. Specificare *"Devices running SQL Server CE"* come tipi di sottoscrittori.
3. Selezioniamo gli oggetti che si vogliono pubblicare: in Fig. 6 possiamo notare che abbiamo selezionato tutte le tabelle e in più la vista *FILES_VIEW* sulla tabella *FILES.* Capiremo perchè abbiamo creato questa vista proseguendo nella trattazione.
4. Nella form successiva mantenere i valori di default.
5. Specificare il nome della pubblicazione.
6. Rispondere positivamente alla domanda se si vogliono creare filtri dinamici per i dati.
7. Specificare che i filtri saranno sulle righe *(horizontally).*
8. Specificare che i filtri saranno dinamici.

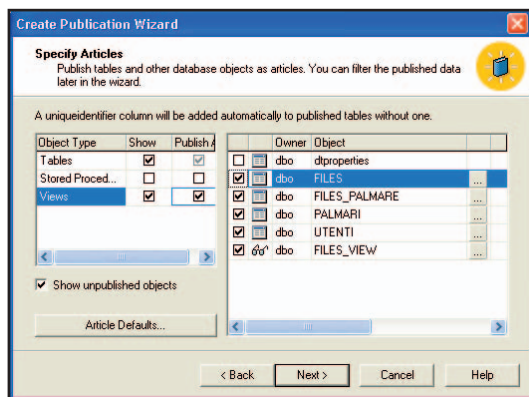



Fig. 6: ***Oggetti selezionati nella pubblicazione.***

9. Specificare i filtri dinamici per ogni tabella.

Ecco i filtri dinamici da specificare per ogni tabella:

Tabella FILES:

```
SELECT <published_columns> FROM [dbo].[FILES] WHERE
COD_FILE IN
   (select COD_FILE_FK FROM FILES_PALMARE INNER JOIN
PALMARI
   ON
PALMARI.COD_PALMARE=FILES_PALMARE.COD_PALMARE_FK
   WHERE DB_USER = SUSER_SNAME())
```

Tabella FILES_PALMARE:

```
SELECT <published_columns> FROM [dbo].[FILES_PALMARE]
WHERE COD_PALMARE_FK IN
( SELECT COD_PALMARE FROM PALMARI W
  WHERE DB_USER = SUSER_SNAME() )
```

Tabella PALMARI:

```
SELECT <published_columns> FROM [dbo].[PALMARI]
                                                WHERE
DB_USER=SUSER_SNAME()
```

Tabella UTENTI:

```
SELECT <published_columns> FROM [dbo].[UTENTI]
                                                WHERE
COD_UTENTE IN (   SELECT COD_UTENTE_FK FROM
PALMARI  WHERE DB_USER=SUSER_SNAME() )
```

Dopo aver inserito i filtri dinamici continuare con i successivi passi del Wizard, mantenendo le impostazioni di default, tranne nell'ultimo passo in cui consigliamo di selezionare il check-box per la creazione immediata dello Snapshot.

CONSIDERAZIONI SUI FILTRI DINAMICI

A questo punto vi dobbiamo alcune spiegazioni sul funzionamento della tecnica dei filtri dinamici offerti da SQL Server. I filtri dinamici permettono di creare delle pubblicazioni con *Merge Replication* e consentono di filtrare i dati delle tabelle pubblicate in funzione del particolare sottoscrittore che accede al Database. I filtri dinamici sono "filtri di riga" e sono applicati su una singola tabella e non su join di tabelle. I benefici nell'utilizzo dei filtri dinamici sono principalmente due:

1. la partizione dei dati (filtrati in funzione dell'utente corrente), permette di utilizzare e gestire una sola pubblicazione dei dati;
2. i sottoscrittori della pubblicazione ricevono solo le informazioni di cui necessitano, poichè i dati sono filtrati in virtù delle proprietà di connessione della sottoscrizione del *Merge Agent*.

In particolare, i dati vengono valutati nel momento in cui parte il processo di Merge che sta replicando i dati tra il client sottoscrittore e il Publisher. Le funzioni di SQL Server più comuni per l'implementazione dei filtri dinamici sono *SUSER_SNAME()* e *HOST_NAME()*. La prima funzione di sistema *SUSER_SNAME()* fornisce lo userid del client sottoscrittore della pubblicazione che sta accedendo al database per effettuare la sincronizzazione dei dati. In particolare, supponiamo che sul pocket PC l'oggetto di merge replication sia stato settato con l'utente "pippo", allora al momento della valutazione del filtro dinamico sulle tabelle di pubblicazione avremo *SUSER_SNAME() = "pippo".* La seconda funzione di sistema *HOST_NAME* mi restituisce il nome del dispositivo o computer sottoscrittore della pubblicazione che si connette al database per la sincronizzazione. Una volta compreso il funzionamento della funzione *SUSER_SNAME,* siamo in grado di comprendere il funzionamento dei filtri dinamici scritti in precedenza. In particolare, l'applicazione del filtro dinamico sulla tabella *UTENTI,* permette di sincronizzare sul Pocket PC solo i dati del proprietario del palmare. Allo stesso modo, il filtro dinamico sulla tabella PALMARI permette di replicare solo le informazioni relative al palmare associato con l'utente corrente. Considerazioni analoghe possono essere fatte anche per gli altri filtri dinamici.

IL TRASFERIMENTO DEI FILE

Prima di tutto dobbiamo spiegare per quale motivo abbiamo creato la vista sui dati della tabella *FILES.* La vista ci è servita per sincronizzare sul Pocket PC le informazioni dei file Multimediali condivisi dalla applicazione *MultimediaServer.* In modo particolare, non abbiamo proiettato i campi *PATH,* e i dati binari che contengono i file veri e propri, ovvero *BINARY_CODE_IMG* e *BINARY_CODE_FILE* per con-

Creare un utente di SQL Server per ogni client PocketPC che accede in sincronizzazione sul Server di Database migliora la sicurezza dei dati delle nostre applicazioni. Infatti, basta disabilitare l'accesso ad un particolare utente per non permettere ad un particolare Pocket PC di accedere ai dati in sincronizzazione.

Fig. 7: Maschera "Inserisci File".

Fig. 9: Lista File scaricati Sul PocketPC.

sentirne un effettivo download solo per effetto dei filtri dinamici. In questo modo, sul palmare avremo sempre la lista aggiornata delle descrizioni dei file disponibili. L'operazione di download effettivo di un file sarà possibile solo dopo una sua selezione sul palmare. Cerchiamo di comprendere il funzionamento tramite un esempio. Supponiamo che un palmare con codice 1 abbia associato l'utente di SQL Server "pippo". Supponiamo, inoltre, che l'utilizzatore del palmare abbia in qualche modo selezionato da un controllo contenente la lista di file disponibili il file *messaggioVocale.mp3* con codice 2; a questo punto bisogna effettuare una operazione di inserimento dei codici precedenti nella tabelle *FILES_PALMARE* sull'istanza locale del database del PocketPC ed eseguire una sincronizzazione dei dati. In particolare dovremo aggiungere nella tabella il record *r1:*

```
(ID_FILES_PALMARE,COD_PALMARE_FK,COD_
                                FILE_FK)(newId(),1,2).  (r1)
```

Le operazioni che saranno eseguite nell'ordine dalla sincronizzazione saranno le seguenti:

1. Sincronizzazione del record *r1* nella tabella *FILES_PALMARE* del Database sul Server.
2. Esecuzione della logica dei filtri dinamici. All'esecuzione del filtro sulla tabella *FILES* troveremo che la valutazione della query seguente per la selezione dei codici dei File da scaricare:

```
(select COD_FILE_FK FROM FILES_PALMARE INNER JOIN
PALMARI
  ON
PALMARI.COD_PALMARE=FILES_PALMARE.COD_
                                          PALMARE_FK
  WHERE DB_USER = SUSER_SNAME())
```

avrà *SUSER_SNAME= "pippo"* e fornirà il codice del file = *2.* Quest'ultimo codice individuerà in maniera univoca il record *f1* della tabella *FILES.*

3. Sincronizzazione del record *f1* individuato nel passo precedente dal filtro dinamico sul database del PocketPC.

Terminata la sincronizzazione il file sarà presente sul palmare e potrà essere processato tramite un programma apposito oppure all'interno del client *MultimediaClient* sul Pocket PC.

INSERIMENTO DI UN FILE PER LA CONDIVISIONE

Nella form principale della maschera iniziale della applicazione *MultimediaServer* possiamo cliccare sul tasto *Inserisci File* per vedere apparire la maschera di Fig. 7. *frmInserisciFile* è la classe relativa a questa Form. È stata inserito un Windows Form Component di tipo *OpenFileDialog* per permettere di ricercare nel File System del computer il file che si vuole inserire nel DataBase; l'operazione di ricerca del file viene attivata con la pressione del tasto con etichetta "...". La selezione del file consentirà di valorizzare in automatico i campi della form: nome, path e tipologia del file. La procedura di salvataggio associata al click del tasto *Salva (btnSalva_ onCLick),* richiamerà la stored procedure *spInserisciFile* di SQL Server per l'inserimento del file specificato. In particolare, la stored procedure riceve, tra gli altri, dei parametri che accettano una rappresentazione binaria del file. La rappresentazione binaria viene ottenuta con il metodo seguente:

```
private static byte[] GetMediaFile(string filePath){
FileStream fs = new FileStream(filePath,
                              FileMode.Open, FileAccess.Read);
BinaryReader br = new BinaryReader(fs);
byte[] mediaObject = br.ReadBytes((int)fs.Length);
br.Close();
fs.Close();
return mediaObject;
}
```

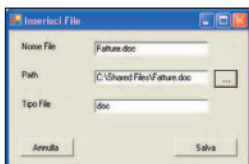
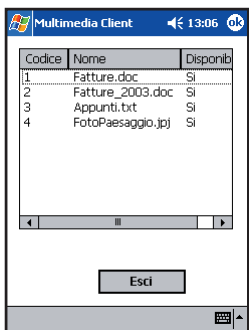
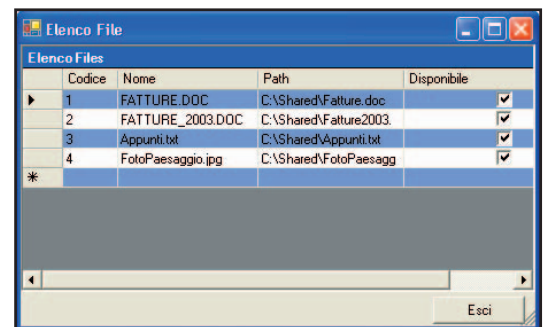



Fig. 8: Lista File disponibili sul Server

Se l'operazione di inserimento è andata a buon fine, avremo memorizzato nel database i file che vogliamo condividere con i client PocketPC. In Fig. 8 possiamo possiamo osservare la maschera che contiene i file inseriti nella nostra istanza di database. Questa maschera è visualizzata con il click sul tasto "Elenco File" della maschera principale. In Fig. 9 possiamo osservare la lista dei file ricevuti dal Pocket PC in seguito ad una operazione di Sincronizzazione. Possiamo vedere come la lista dei file coincida con la quelli inseriti sul Server. Per qualunque altra considerazione di carattere esclusivamente programmativo lasciamo che il lettore esplori il codice in allegato al CD di "ioProgrammo" rimanendo comunque a disposizione sul *FORUM* di *ioprogrammo.it* per qualunque delucidazione sull'intera applicazione.

Elmiro Tavolaro

Zip: creazione ed estrazione di file

Un WinZip con Java

parte terza

Questo mese ci dedicheremo alla realizzazione delle routine indispensabili per consentire all'utente di estrarre uno o più file dall'interno di un archivio ZIP. Senza di esse nessun programma di questo tipo avrebbe senso!

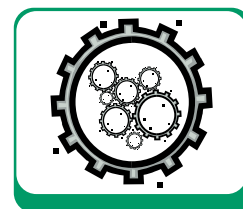
Stiamo sviluppando un'applicazione chiamata *SwingZIP*, una sorta di WinZip multipiattaforma. Il compito di questo software è consentire all'utente la gestione degli archivi compressi in formato ZIP, fornendo ogni funzione di base attraverso un'intuitiva interfaccia grafica. Quando il programma sarà completo, l'utente potrà usarlo per creare, aggiornare ed estrarre i comuni archivi ZIP. Al momento, grazie a quanto realizzato nel corso della precedente parte del tutorial, SwingZIP può aprire e mostrare il contenuto di un archivio, ma non è ancora in grado di fare altro. Questo mese insegneremo alla nostra applicazione come fare per estrarre totalmente o parzialmente un archivio.

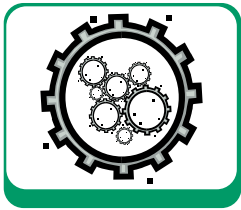
ESTRAZIONE DI UN ARCHIVIO

L'interfaccia del programma, già completa, propone un pulsante etichettato "*Estrai*", che può essere utilizzato dopo aver aperto un archivio ZIP. Quando l'utente preme questo bottone, il codice sottostante richiama il metodo privato *zipExtract()*. Al momento, il corpo del metodo è vuoto. Dobbiamo popolarlo con il codice necessario per gestire ed eseguire l'operazione di estrazione:

```
private void zipExtract() {
  // Dove vuole estrarre l'utente?
  JFileChooser fileChooser = new JFileChooser();
  fileChooser.setFileSelectionMode(
                          JFileChooser.DIRECTORIES_ONLY);
  int option = fileChooser.showSaveDialog(this);
  if (option != JFileChooser.APPROVE_OPTION) return;
  final File destination = fileChooser.getSelectedFile();
  // Ci sono elementi selezionati nella lista?
  final boolean extractAll;
  if (!filesList.isSelectionEmpty()) {
    // Chiede all'utente se vuole estrarre solo i
                                              selezionati.
    String[] options = {
      "Solo i file selezionati", "Tutto l'archivio" };
    String selected = (String)
    JOptionPane.showInputDialog(this, "Quali file devo
      estrarre?", "Richiesta", JOptionPane.QUESTION_
                   MESSAGE, null, options, options[0] );
    if (selected == null) return; // Azione annullata.
    extractAll = (selected == options[1]);
  } else extractAll = true;
  // Si rende inattiva l'interfaccia grafica.
  b1.setEnabled(false);
  b2.setEnabled(false);
  b3.setEnabled(false);
  b4.setEnabled(false);
  b5.setEnabled(false);
  …
      // Riabilita l'interfaccia.
      b1.setEnabled(true);
      b2.setEnabled(true);
      b3.setEnabled(true);
      b4.setEnabled(true);
      b5.setEnabled(true); }
  });
  thread.start();}
```

Per prima cosa, è necessario interpellare l'utente. Bisogna estrarre dei file, va bene, ma in quale directory dovrà essere riposto il risultato dell'operazione? Spetta all'utente deciderlo. Facciamo ricorso ad un oggetto *JFileChooser*:

```
JFileChooser fileChooser = new JFileChooser();
fileChooser.setFileSelectionMode(
                          JFileChooser.DIRECTORIES_ONLY);
int option = fileChooser.showSaveDialog(this);
```

ARCHIVI ZIP

Java fornisce delle librerie che consentono la totale astrazione dal tipo di compressione impiegata dal formato ZIP. Infatti, utilizzando le API del package *java.util.zip*, non vi ritroverete mai ad avere a che fare con i complicati algoritmi che sono alla base della compressione dei file. Le classi offerte dalla libreria di Java fanno tutto da sole. Tuttavia, se vi interessa scoprire come funzioni effettivamente la compressione ZIP, magari per scrivere le vostre API personali, cominciate dando uno sguardo all'indirizzo Web: http://www.pkware.com/products/enterprise/white_papers/appnote.html

```
if (option != JFileChooser.APPROVE_OPTION) return;
final File destination = fileChooser.getSelectedFile();
```

Il file chooser, in questo caso, deve permettere la selezione di una directory, e non di un file come avviene di solito. Pertanto, invece di ricorrere allo *ZIPFileFilter* utilizzato in precedenza nel metodo *zipOpen()*, è sufficiente impostare il tipo di selezione sulla costante *JFileChooser.DIRECTORIES_ONLY*, con il metodo *setFileSelectionMode()*. Dopo aver mostrato la finestra di salvataggio con *showSaveDialog()* (Fig. 1), è necessario controllare che l'operazione non sia stata annullata. Se l'utente ha compiuto la propria scelta in maniera regolare, senza comandare un annullamento, il percorso selezionato deve essere recuperato e memorizzato in una variabile di tipo File. Questa variabile è final, per motivi che già abbiamo discusso nel corso della parte precedente, mentre allestivamo il metodo *zipOpen()*. Anche in questo caso, infatti, le operazioni più pesanti verranno lanciate dall'interno di un thread secondario, rappresentato da una inner-class innestata nel codice del metodo. Se il riferimento al percorso selezionato non fosse final, una classe interna non potrebbe accedervi.

Fig. 1: **SwingZIP chiede all'utente in quale directory desidera decomprimere i file.**

Prima di procedere con l'estrazione, ad ogni modo, bisogna compiere un'ulteriore verifica. Mettiamo il caso che l'utente abbia selezionato, nella *JList* centrale, un sottoinsieme dei file compresi nell'archivio. In questa situazione, dobbiamo estrarre tutto l'archivio o solamente la parte selezionata? Facciamolo scegliere all'utente (Fig. 2):

Fig. 2: **Se nella lista dei file compressi è stata effettuata una selezione che abbraccia una o più entry, SwingZIP chiede all'utente se desidera estrarre solo gli elementi selezionati oppure l'intero archivio.**

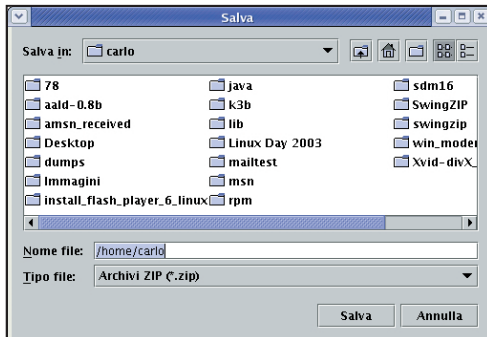
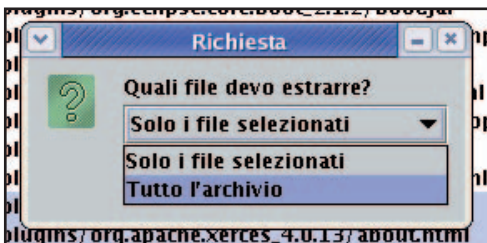
```
final boolean extractAll;
if (!filesList.isSelectionEmpty())
{
  // Chiede all'utente se vuole estrarre solo i selezionati.
  String[] options =
  { "Solo i file selezionati", "Tutto l'archivio" };
  String selected = (String)JOptionPane.showInputDialog(
          this, "Quali file devo estrarre?", "Richiesta",
                JOptionPane.QUESTION_MESSAGE,
                      null, options, options[0] );
  if (selected == null) return; // Azione annullata.
    extractAll = (selected == options[1]);
}
else extractAll = true;
```

Al termine di questa routine, il booleano *extractAll* sarà *true* se nessuna selezione è stata effettuata oppure se, in presenza di una selezione, l'utente ha scelto di estrarre completamente l'archivio. In altri casi il valore sarà ovviamente *false*. Anche *extractAll* è una variabile *final*: il codice che sarà eseguito nel thread secondario, infatti, dovrà accedervi. Ora è possibile disattivare l'interfaccia grafica, per evitare interferenze da parte dell'utente, e lanciare il thread secondario contenente la vera e propria routine di estrazione:

```
// Si rende inattiva l'interfaccia grafica.
b1.setEnabled(false);
b2.setEnabled(false);
b3.setEnabled(false);
b4.setEnabled(false);
b5.setEnabled(false);
// Memorizza un alias della finestra, sostitutivo di this nella
// classe interna che sta per essere creata.
final SwingZIP myself = this;
// Passa all'esecuzione su thread secondario.
Thread thread = new Thread(new Runnable()
{ public void run() {// Routine di estrazione. }}
thread.start();
```

Prima della dichiarazione e del lancio del nuovo thread si salva un riferimento all'oggetto corrente nella variabile finale *myself.* Dall'interno della inner-class del thread secondario, infatti, la parola chiave *this* non fornirà più un riferimento all'applicazione in sé, ma all'oggetto *Runnable* contenuto nel *thread* secondario. Siccome il riferimento alla finestra principale ci è utile, è meglio conservarlo di nostra iniziativa.

Passiamo all'analisi della vera e propria routine di estrazione. Bisogna aprire l'archivio ZIP corrente, in modo da avere accesso al suo contenuto:

```
ZipFile zipFile = null;
try {
  zipFile = new ZipFile(file);
}
catch (IOException e)
{  // In caso di errore durante l'apertura.
  showMessage("Errore!", "Impossibile aprire l'archivio",
                    JOptionPane.ERROR_MESSAGE);
  // Riabilita l'interfaccia.
  b1.setEnabled(true);
  b2.setEnabled(true);
  b3.setEnabled(true);
  b4.setEnabled(true);
  b5.setEnabled(true);
  // Abbandona l'estrazione.
  return;
}
```

L'operazione è a rischio, l'apertura potrebbe fallire. Ad esempio, il file potrebbe essere divenuto inaccessibile solo qualche attimo prima che l'utente comandasse l'estrazione. Meglio usare un blocco *try... catch*. In caso di eccezione l'utente è informato dell'accaduto, l'interfaccia viene riabilitata e la routine di estrazione abbandonata. Se si supera questo blocco *try... catch*, tutto è andato a buon fine, e la variabile *zipFile* contiene un riferimento valido all'archivio da estrarre. Il passo successivo è compilare una lista dei file, o per meglio dire delle *entry*, da estrarre:

```
Object[] entries = null;
if (!extractAll) {
   // Se bisogna estrarre solo una parte delle entry.
   entries = filesList.getSelectedValues(); }
else {
   // Se bisogna estrarre tutte le entry.
   entries = new ZipEntry[zipFile.size()];
   Enumeration entriesEnumeration = zipFile.entries();
   for (int i = 0; i < entries.length; i++)
   { entries[i] = entriesEnumeration.nextElement(); }
}
```

Se *extractAll* è *false* c'è una selezione in corso, e solo il contenuto di tale selezione deve essere estratto. In caso contrario, bisogna generare la lista di tutte le entry dell'archivio. Nella parte bassa della finestra dell'applicazione abbiamo inserito una barra di progresso. Siccome non ce l'abbiamo messa per bellezza, l'aggiorneremo frequentemente durante l'estrazione, in modo che l'utente possa capire che l'applicazione è a lavoro, stimando anche quanto tempo dovrà ancora attendere. Per far questo dovremo, di volta in volta, calcolare la percentuale completata, con una semplice proporzione, ma non sarebbe possibile farlo senza sapere a priori quanto sono grandi le entry da estrarre:

```
long totalSize = 0;
for (int i = 0; i < entries.length; i++)
{
   long size = ((ZipEntry)entries[i]).getSize();
   if (size > -1) totalSize += size;
}
```

Se in *totalSize* memorizziamo la dimensione totale delle entry da estrarre, un'altra variabile dello stesso tipo dovrà essere impiegata per annotare quanto è stato già estratto:

```
long doneBytes = 0;
```

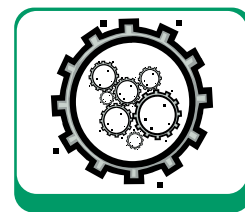
L'estrazione è un'operazione che può fallire totalmente o parzialmente. Se fallisce totalmente, è chiaro che nessun file è stato estratto. Ad ogni modo può capitare che alcune entry possano essere estratte ed altre no. Un fallimento parziale, appunto. Mettiamo il caso che una entry debba andare a sovrascrivere un file già esistente, e che proprio questo file non possa essere sovrascritto per mancanza dei permessi necessari.

Sarebbe d'uopo informare l'utente circa le entry che non è stato possibile estrarre. Per questo motivo si dichiara un *Vector* che conterrà, sotto forma di stringhe, tutti gli avvisi da presentare all'utente:

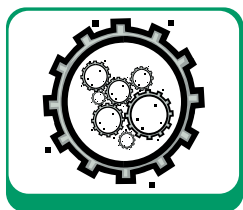
```
Vector log = new Vector();
```

Ora che tutto il necessario è stato predisposto, possiamo cominciare ad estrarre le *entry*. Dentro l'array *entries*, lo ricordo, conserviamo i riferimenti verso tutte le *entry* che dobbiamo considerare. Non ci resta che ciclare attraverso questo array, considerando ogni volta un suo diverso elemento:

```
for (int i = 0; i < entries.length; i++)
{
   // Considera l'entry corrente.
   ZipEntry entry = (ZipEntry)entries[i];
   // Quanto pesa l'entry in considerazione?
   long entrySize = entry.getSize();
   // Elabora il percorso sul disco per l'estrazione.
   File aux = new File(destination, entry.getName());
   // Dichiara i riferimenti per l'estrazione.
   InputStream in = null;
   FileOutputStream out = null;
   // Segue l'estrazione dell'entry.
   // ...
}
```

Dentro questo ciclo *for*, quindi, recuperiamo l'*entry* attualmente in considerazione e ne annotiamo la dimensione (sarà utile per calcolare la percentuale di completamento dell'operazione). Nel riferimento *aux*, di tipo *File*, abbiamo memorizzato il punto del file system in cui dovrà essere estratta la entry. Questo percorso si ottiene unendo la directory selezionata dall'utente con il nome della entry. Il nome di una entry può contenere un percorso relativo. Meglio supporre un esempio: l'utente ha deciso di estrarre l'archivio nella directory */home/carlo*, e l'archivio contiene una entry chiamata */documenti/curriculum.txt.* Al termine dell'estrazione, il file corrispondente sarà al percorso */home/carlo/documenti/curriculum.txt.* Se la directory */home/carlo/documenti* non esiste, SwingZIP dovrà provvedere alla sua creazione. Prima di procedere con l'estrazione, tornando alla valutazione del codice compreso nel ciclo for, vengono dichiarati i riferimenti verso i canali di input e di output che serviranno per leggere dall'archivio e per scrivere sul file system. Andiamo oltre. Dentro il ciclo *for*, ora, serve un blocco *try... catch*, poiché si entra nella fase rischiosa dell'operazione:

PHILLIP W. KATZ

Phillip W. Katz è il "padre" del formato ZIP. La sua storica creazione fu PKZIP (Phillip Katz ZIP, per l'appunto), il primo programma della storia capace di trattare gli archivi ZIP nella forma in cui li conosciamo oggi. Phillip è morto nell'Aprile 2000, all'età di 37 anni.

http://webnews.html.it/storia/58.htm

```
try  {
    if (entry.isDirectory()) {
        // Se l'entry è una directory, la crea.
        aux.mkdirs(); }
    else  {
        // Se l'entry è un file, crea tutte le directory
        // del suo percorso.
        aux.getParentFile().mkdirs();
        // Apre i canali di comunicazione.
        in = zipFile.getInputStream(entry);
        out = new FileOutputStream(aux);
        // Prepara il buffer per la copia.
        byte[] buffer = new byte[1024];
        long doneEntryBytes = 0;
        int l;
            // Esegue il trasferimento dei dati.
            while ((l = in.read(buffer, 0, buffer.length)) != -1)
            { out.write(buffer, 0, l);
              // Aggiorna la barra di progresso.
              doneEntryBytes += l;
              progressBar.setValue((int)Math.round(((doneBytes
                  + doneEntryBytes) * 100D) /totalSize)); }
    } }
catch (IOException e)  {
    // L'estrazione dell'entry corrente non è riuscita.
    log.add("Estrazione di " + entry.getName() + " fallita");
}
finally {
    // Chiusura dei canali di comunicazione.
    if (out != null) try
    {out.close();}
    catch (Exception e) {}
    if (in != null) try
    { in.close(); }
    catch (Exception e) {}
    // Aggiornamento della barra di progresso.
    doneBytes += entrySize;
    progressBar.setValue((int)Math.round(
                    (doneBytes * 100D) / totalSize));
}
```

Si distingue il caso in cui l'entry considerata sia una directory oppure un file. Nel primo caso non resta che riprodurre la directory sul file system, usando il metodo *mkdirs()* degli oggetti *File*. Nel secondo caso, invece, si procedere con l'estrazione. Nel caso il percorso del file contenga delle directory che non sono ancora state create, si provvede con una chiamata a *mkdirs()*. Dopo aver accertato che il percorso cui sarà posizionato il file è valido ed esistente, i canali vengono aperti e le operazioni di lettura e scrittura eseguite secondo le stesse norme che abbiamo osservato nel corso della prima parte di questo tutorial. L'unica novità è rappresentata dalla barra di progresso. Nel caso intervenga qualche eccezione nel corso delle operazioni di lettura e scrittura, il blocco catch provvederà ad annotare il problema nel vettore dei log. Il blocco finally, che è sempre e comunque eseguito subito prima dell'abbandono della struttura, chiude i canali eventualmente aperti ed aggiorna la barra di progresso considerando completamente valutata l'entry corrente. Dopo il ciclo for che cura l'estrazione di ogni entry scelta, non resta che eseguire alcune procedure finali. La barra di progresso viene resettata:

```
progressBar.setValue(0);
```

Se c'è qualcosa nel vettore *log* bisogna informare l'utente circa i problemi riscontrati. Si genera al volo una finestra di dialogo modale che elenca tutto il contenuto del log:

```
if (log.size() != 0) {
    // Crea una finestra di dialogo per visualizzare il log.
    JDialog logDialog = new JDialog(
                myself, "SwingZIP Extraction Log", true);
    JList logList = new JList(log);
    JScrollPane scrollPane = new JScrollPane(logList);
    scrollPane.setPreferredSize(new Dimension(400, 400));
    JPanel content = new JPanel(new BorderLayout(3, 3));
    content.setBorder(new EmptyBorder(3, 3, 3, 3));
    content.add(new JLabel("Sono stati riscontrati
                    i seguenti problemi", JLabel.CENTER
                            ), BorderLayout.NORTH);
    content.add(scrollPane, BorderLayout.CENTER);
    logDialog.getContentPane().add(content);
    logDialog.pack();
    logDialog.setLocationRelativeTo(myself);
    logDialog.show();
}
```

Infine, prima di abbandonare il thread secondario, l'interfaccia viene riabilitata:

```
b1.setEnabled(true);
b2.setEnabled(true);
b3.setEnabled(true);
b4.setEnabled(true);
b5.setEnabled(true);
```

Non ci resta che ricompilare i sorgenti e fare qualche test.

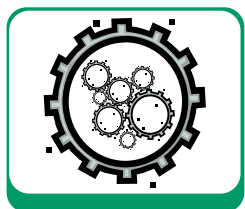
IL MESE PROSSIMO...

Lo spazio a nostra disposizione, per questo mese, è terminato. Ci ritroveremo su queste pagine tra trenta giorni circa, per la quarta parte del tutorial. Il prossimo appuntamento sarà dedicato alla stesura del metodo *zipDelete()*, utile per rimuovere uno o più file dall'interno di un archivio ZIP esistente. Non mancate.

Carlo Pelliccia

Introduzione alla programmazione di Office 2003

Supporto del formato XML in Office 2003

Con questo appuntamento, dedicato al formato XML, iniziamo l'esplorazione delle features di Office 2003 che lo configurano come una vera e propria piattaforma di sviluppo.

Microsoft Office 2003, la nuova release della famiglia di applicazioni dedicata agli information Worker, ha importanti innovazioni che la rendono più produttiva, in numerosi campi, ed aperta verso altre applicazioni e sistemi. Office 2003, infatti, grazie al supporto della tecnologia XML consente di sviluppare delle soluzioni robuste, intelligenti ed in grado di gestire informazioni che finora erano racchiuse in pesanti file binari. L'XML, in Office 2003, contribuisce alla giusta separazione tra regole di business, dati, e presentazione. Per esempio, con Microsoft Excel 2003 si può continuare a creare dei documenti con formule e formattazione tipica delle applicazioni Desktop ed inserire i "dati sensibili" in dei tag XML per poi esportarli verso database oppure pubblicarli sul Web. Questo è reso possibile dal supporto di strumenti come gli schemi XML (XSD - *XML Schema Definition*), le trasformazioni XSL (*Extensible Stylesheet Language*) e con l'introduzione degli *Smart Document*. Questi ultimi sono dei "documenti intelligenti" che consentono di unire i vantaggi delle applicazioni desktop a quelli della tecnologia XML. Gli schemi o modelli XML, invece, sono una sorta di modello per i dati XML (più o meno come il modello Entità Relazione lo è per i dati dei database relazionali) e servono per creare e convalidare il contenuto dei documenti XML. I file XSL sono degli strumenti di interpretazione che consentono di trasformare i dati presenti in un file XML in un formato più adatto alla pubblicazione sul Web o alla esportazione su altri sistemi o applicazioni. In questo articolo avvieremo l'implementazione di un sistema integrato per gestire i curriculum vitae dei neo laureati. I dati dei curriculum vengono prima inseriti in un documento XML-Word (opportunamente formattato e che ha associato uno schema XSD personalizzato) e poi attraverso una trasformazione XSL vengono archiviati in dei file XML che possono essere importati, facilmente, in una o più tabelle Access, pubblicati su Internet o interrogati con delle query. Quindi, per realizzare questo sistema, abbiamo bisogno di uno schema XSD, di una o più trasformazioni XSL e di alcune procedure VBA di supporto. Naturalmente, nei nostri articoli, dei linguaggi XML, XSL e XSD illustreremo soltanto gli elementi che man mano utilizzeremo.

• processore Pentium 233 MHz o superiore; • sistema operativo Windows 2000 Service Pack 3 o successivo, Windows XP o successivo; • 64 MB di memoria RAM minimo, • 128 MB consigliata; • lo spazio richiesto sul disco rigido va da 245 MB a 2 GB dipende dal tipo di installazione e dalla edizione di Office 2003.

XML E SMART DOCUMENT

Descriviamo sommariamente quali sono le nuove caratteristiche di Office 2003 introdotte con il supporto dell'XML. In Excel, in realtà, già nella versione XP era presente il formato nativo XMLSS (*XML Spreadsheep Schema*) che ora è stato esteso con il supporto degli schemi XSD e con l'introduzione dello strumento grafico *Mapping XML*, che consente di migliorare la fase d'importazione e creazione dei documenti XML. In Word 2003, invece, è stato introdotto il formato WordML - che permette di avere un file XML valido senza perdere la formattazione applicata in Word - e il supporto degli schemi XSD e delle trasformazioni XSL. In Word per la gestione dei Tag XML è prevista, nel riquadro attività, la scheda struttura XML, in essa sono presenti i comandi per creare e controllare i documenti XML. Il riquadro attività può essere selezionato attraverso il menu *Visualizza* o con la seguente riga di codice:

```
Application.TaskPanes(
        Index:=wdTaskPaneXMLStructure).Visible = True
```

Con gli *Smart Document*, utilizzabili sia in Word 2003 che in Excel 2003, gli *Smart Tag* sono stati integrati con gli elementi XML. Gli *Smart Document* sono dei documenti che, come gli *Smart Tag*, forniscono suggerimenti all'utilizzatore. Essi vengono creati attraverso lo *Smart Document Software Development Kit* ed associati ai documenti attraverso i pacchetti di espansione XML; questi sono degli insiemi di file gestiti da un file di nome *manifest.xml*. Gli *Smart Document* forniscono i suggerimenti attraverso delle schede programmabili mostrati sul

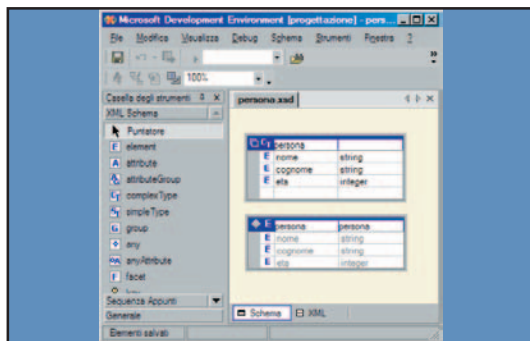



Fig. 1: *Lo schema XML* **persona** *costruito con Visual Studio .Net.*

riquadro attività che così diventa una vera e propria maschera anche collegabile ad un database locale o in rete (ad un Web Service). Per aggiungere un'espansione XML bisogna scegliere la voce *"Modelli e aggiunte"* dal menu *strumenti* e poi selezionare la scheda *Pacchetti di espansione XML*. Lo *Smart Document Software Development Kit* può essere scaricato dal sito della Microsoft.

SCHEMA XML IN WORD

Uno schema XML in generale è un file con estensione XSD, scritto nel linguaggio XSD (con sintassi XML), e serve a progettare la struttura di documenti XML. In uno schema è possibile definire i nomi degli elementi XML, i loro attributi ed i dati formattati, inoltre è possibile stabilire come gli elementi XML devono essere organizzati e quali dati sono consentiti. Gli schemi XSD, tra l'altro, vengono utilizzati per mantenere la coerenza tra i dati di diverse organizzazioni o sistemi; infatti, nelle loro comunicazioni generano dei messaggi in formato XML attraverso lo stesso schema (per esempio nel settore sanitario si usa lo schema *H7*). Un elemento importante nello schema XML è lo spazio dei nomi che è un identificatore univoco per gli elementi presenti nel documento XML. Questo evita conflitti per esempio nel caso in cui si definisca un elemento con lo stesso nome di uno dello schema di Word. In Word, a un documento che ha associato uno schema personale, durante la fase di salvataggio, è associato lo schema *WordML* così da preservarne la formattazione; però si può anche decidere di salvare soltanto i dati del documento e quindi perdere la formattazione definita in Word (controllate la Fig. 5). È anche possibile salvare i dati XML in un documento *.doc* o in un modello *.dot,* in questi casi però i dati XML possono essere letti ed interpretati solo in Word. Gli schemi associati ai documenti sono contenuti nel raccoglitore di schemi di Word selezionabile attraverso il menu *strumenti / modelli ed aggiunte.* Ora, prima di presentare lo schema XSD del nostro sistema con un semplice esempio presentiamo come si implementa uno schema XML.

IL NOSTRO PRIMO SCHEMA XML

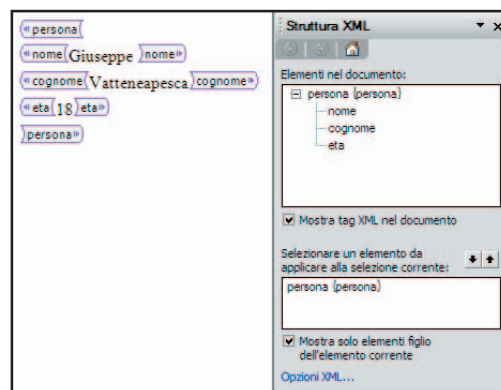
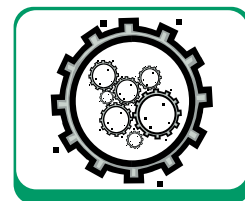
Creiamo lo schema XML per definire dei documenti XML che contengono il nome, il cognome e l'età delle persone. Uno schema XML può essere sviluppato direttamente, inserendo le istruzioni XSD in un file testo con estensione *XSD* oppure attraverso un tool grafico, per esempio attraverso Visual Studio .Net (si controlli la Fig. 1). Uno schema XML è costituito dall'elemento schema di primo livello che include la definizione degli spazi dei nomi, quindi possiamo scrivere:

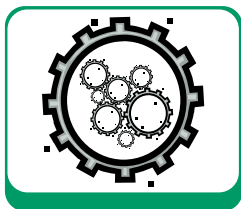
```
<xs:schema xmlns:xs=
                    "http://www.w3.org/2001/XMLSchema"
xmlns="persona" targetNamespace="persona">
```

La dichiarazione *xs:schema* indica che si tratta di un elemento schema e che *xs*: sarà il prefisso degli elementi dello schema, la dichiarazione successiva indica che tutti i tag devono essere interpretati in base allo spazio dei nomi stabilito dal W3C nel 2001. Gli altri due attributi danno allo schema il nome persona. Nell'elemento schema è possibile definire dei tipi complessi o semplici (elementi *simpleType* e *complexType*) e dichiarare attributi (*attribute*) ed elementi *(element)*. Gli elementi, dichiarati come *complexType,* possono contenere attributi ed elementi (i *simpleType* invece non possono contenere elementi). La dichiarazione di un *complexType* formato da tre elementi (nome, cognome ed età) di tipo primitivo (cioè *string, integer*) è la seguente:

Fig. 2: *La scheda Struttura XML.*

```
<xs:complexType name="persona">
  <xs:sequence>
    <xs:element name="nome" type="xs:string" />
    <xs:element name="cognome" type="xs:string" />
    <xs:element name="eta" type="xs:integer" />
  </xs:sequence>
</xs:complexType>
```

Nel codice XSD precedente facciamo notare che la dichiarazione di un elemento, fatta attraverso l'attributo *type,* associa un nome ad una definizione di tipo, questo in generale può essere un tipo primitivo, un *simpleType* o un *complexType*. Nella definizione del tipo complesso, inoltre, viene utilizzato l'elemento *sequence* per specificare che gli elementi nel documento XML dovranno essere riportati nella sequenza specificata all'interno del tipo complesso.

OFFICE 2004

Le edizioni di Office 2003 sono 4 e nessuna di esse gira sui sistemi operativi di vecchia generazione (Windows 95/98/ME/NT). Inoltre le caratteristiche XML avanzate, cioè la gestione degli schemi XSD e dei Smart Document sono supportati solo dalla versione Professional di Office 2003 e dalla versione autonoma di Word 2003.

GLOSSARIO

XML

XML è l'acronimo di *eXtensible Markup Language* (Linguaggio di Marcatura Estendibile). XML è un meta-linguaggio che consente di manipolare dati strutturati usando file di testo. L'obiettivo principale dell'XML è quello di favorire la portabilità dei dati tra diverse piattaforme. Un documento XML è costituito da Tag strutturati che racchiudono dati. Il documento è organizzato secondo una struttura ad albero (*Treelike*) con una radice (*root*) e tanti nodi figli (*child*).

NOTA

IL PARSER MSXML

Se cliccate sul file con estensione XML (che non ha associato lo schema WordWL) come risultato otterrete l'avvio di Internet Explorer e l'interpretazione del contenuto del file. Il file è interpretato da MSXML (Microsoft XML Parser) una libreria presente in Internet Explorer già dalla versione 4. Il Parser in parole povere è un software che legge un documento XML lo interpreta e permette l'accesso al suo contenuto.

Per completare il nostro esempio dobbiamo dichiarare un elemento del tipo complesso e chiudere lo schema:

```
<xs:element name="persona"
                              type="persona"></xs:element>
</xs:schema>
```

Copiando, in un file XSD, il codice presentato otteniamo lo schema che permette di gestire, semplici, documenti XML come il seguente.

```
<persona>
<nome>Giuseppe<nome>
<cognome>Vatteneapesca<cognome>
<eta>18<eta>
<persona>
```

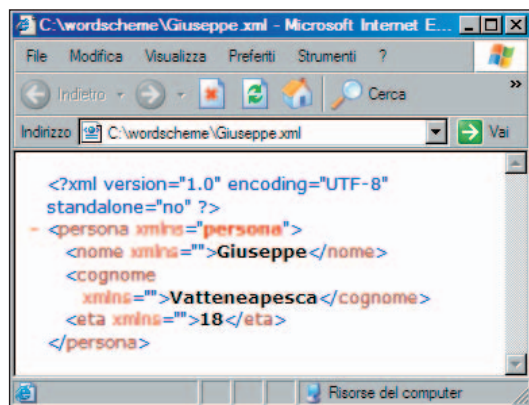



Fig. 3: **Il file XML prodotto da Word con salva solo dati.**

CARICHIAMO LO SCHEMA

Per caricare uno schema XML in un documento Word possiamo procedere nel seguente modo:

1. visualizzare il *riquadro attività* e su di esso selezionare la scheda *Struttura XML*;
2. selezionare *"Modelli e Aggiunte"* e proseguire con la selezione del file XSD che contiene lo schema (nel nostro caso il file *persona.xsd*);
3. nella maschera, *impostazioni schemi*, che compare dopo aver selezionato il file XSD, specificare un alias per il nome del file (si veda la Fig. 4).

Dopo che al documento è stato associato lo schema, con gli strumenti della scheda *Struttura XML* è possibile, attraverso semplici clic, inserire i Tag XML, nel documento. In alternativa lo schema *persona.xsd* può essere associato al documento attraverso la seguente macro VBA.

```
Sub installaschemaxsd()
 ActiveDocument.XMLSchemaReferences.Add _
 "persona", _
 "persona", "C:\Wordscheme\persona.xsd"
End Sub
```

Dove *XMLSchemaReferences* rappresenta la collezione di schemi associati al documento. La sintassi della *Add* (che permette di inserire uno schema nella collezione) è la seguente:

```
Add(NamespaceURI, Alias, FileName, InstallForAllUsers)
```

NamespaceURI è il nome dello schema come definito nel file XSD, gli altri parametri sono l'*Alias*, il nome del file e un parametro *Boolean* che specifica se tutti gli utenti possono accedere allo schema. Facciamo notare che tutti i parametri sono facoltativi.

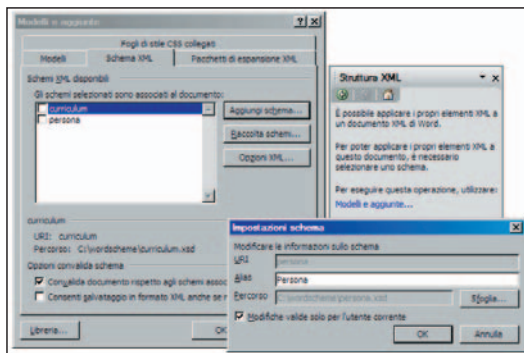
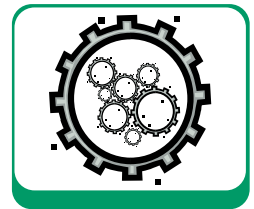



Fig. 4: **La fase di caricamento dello schema XSD persona.**

CONVALIDA DEI DOCUMENTI XML

Sulla scheda *Struttura XML*, dopo aver caricato lo schema, compare il collegamento *Opzioni XML* con il quale è possibile aprire la finestra che permette d'impostare le *Opzioni di salvataggio*, di *convalida* e di *visualizzazione* XML. In particolare le opzioni di convalida permettono di stabilire se "convalidare il contenuto dei documenti XML in base agli schemi XML specificati" oppure "nascondere le violazioni dello schema", ecc. Se selezionate la prima opzione e non la seconda, man mano che inserite i Tag XML nel documento, potete notare che nel controllo *Elementi del Documento* (che mostra la gerarchia XML) della scheda *Struttura XML* sono mostrati, con dei simboli ad hoc, quali nodi non sono convalidati in base allo schema. Inoltre, quando si avvicina il mouse a un nodo non convalidato, viene mostrato un messaggio con la descrizione dell'errore.
Questo può essere fatto anche da programma con la seguente macro VBA che utilizza il modello di oggetti Word.

```
Sub validate()
 Dim Nodo As XMLNode
 For Each Nodo In ActiveDocument.XMLNodes
```

```
    Nodo.validate
    If Nodo.ValidationStatus <> wdXMLValidationStatusOK Then
      MsgBox "errore su: " + Nodo.BaseName _
      & " - " + Nodo.ValidationErrorText
    End If
  Next
End Sub
```

XMLNode rappresenta un singolo elemento XML applicato al documento (*XMLNodes* è l'insieme di questi nodi); la proprietà *ValidationStatus* permette di rilevare lo stato del nodo in base allo schema associato, cioè stabilisce se rispetta l'ordine d'inserimento e se il dato in esso specificato è quello richiesto dallo schema ecc. Se si verificano degli errori questi sono mostrati attraverso la *ValidationErrorText*. Ora definiamo lo schema XML per i documenti che conterranno i dati dei curriculum, nell'esempio ipotizziamo che ai curriculum siano aggiunti la data di presentazione, un numero di protocollo (che è univoco) e delle note. Nel successivo appuntamento descriveremo come impostare ed utilizzare questi tre valori.

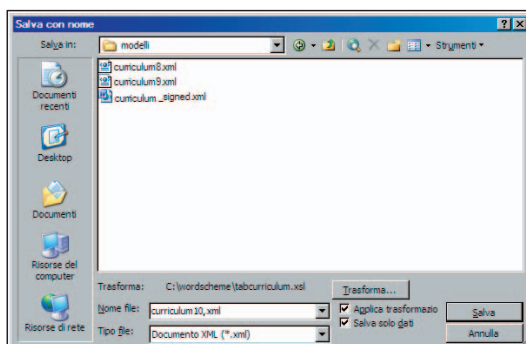



Fig. 5: ***La maschera di Word che permette di salvare il documento utilizzando una trasformazione XSL.***

LO SCHEMA PER I CURRICULUM

In generale, nel curriculum si specificano informazioni come: dati anagrafici, indirizzi, titoli di studio, aspettative professionali, ecc. Per semplificare, nel nostro sistema prevediamo dei documenti in cui vengono specificati soltanto: dati anagrafici, indirizzo, dati laurea e i dati per catalogarli *(datarichiesta, numeroprotocollo, note)*. Naturalmente il sistema è facilmente ampliabile. Lo schema XSD che permette di gestire queste informazioni nei documenti XML è il seguente.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
                                    xmlns="curriculum"
targetNamespace="curriculum">
<xs:complexType name="curriculum">
  <xs:sequence>
    <xs:element name="numprotocollo" type="xs:string">
```

```
                                          </xs:element>
    <xs:element name="datarichiesta" type="xs:string" />
    <xs:element name="persona" type="persona" />
    <xs:element name="indirizzo" type="indirizzo" />
    <xs:element name="laurea" type="laurea" />
    <xs:element name="note" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="persona">
  <xs:sequence>
    <xs:element name="nome" type="xs:string" />
    <xs:element name="cognome" type="xs:string" />
    <xs:element name="eta" type="xs:integer" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="indirizzo">
  <xs:sequence>
    <xs:element name="via" type="xs:string" />
    <xs:element name="cap" type="xs:string" />
    <xs:element name="citta" type="xs:string" />
    <xs:element name="provincia" type="xs:string" />
  </xs:sequence>
</xs:complexType>
<xs:complexType name="laurea">
  <xs:sequence>
    <xs:element name="universita" type="xs:string" />
    <xs:element name="tipolaurea" type="xs:string" />
    <xs:element name="specializzazione"
                                  type="xs:string" />
    <xs:element name="voto" type="xs:integer" />
  </xs:sequence>
</xs:complexType>
<xs:element name="curriculum" type="curriculum">
                                          </xs:element>
</xs:schema>
```

Lo schema precedente è composto dai seguenti *complexType: curriculum, persona, indirizzo* e *laurea.* Questi sono raggruppati nel tipo complesso *curriculum.* Quest'ultimo presenta altri tre elementi: *numprotocollo, datarichiesta* e *note. Curriculum,* oltre a raggruppare gli elementi, ne fissa l'ordine attraverso l'elemento *sequence.*

CONCLUSIONI

In questo articolo abbiamo presentato le caratteristiche XML principali di Office 2003 e contemporaneamente abbiamo spiegato come costruire uno schema XSD. Questo schema fa parte del sistema che permette di archiviare i curriculum vitae dei neo laureati. Nel prossimo articolo completeremo l'applicazione e parallelamente introdurremo nuovi concetti sul vasto universo della tecnologia XML, in particolare vedremo come rappresentare ed esportare i dati XML attraverso i file XSL …

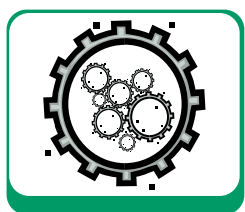
Massimo Autiero

W3C

Per informazioni sullo sviluppo dello standard XML consultate http://www.w3.org/xml che è il sito ufficiale del Worldwide Web Consortium (W3C) il consorzio che promuove questo ed altri standard.

I concetti alla base dei linguaggi di programmazione

Realizzare uno Scripting Engine

Supportare linguaggi di programmazione tramite scripting engine può essere l'idea vincente per garantire un futuro al proprio software. In questo articolo vedremo come concretizzare questa idea.

Sono molte le applicazioni di oggi che consentono di essere programmare per garantire versatilità e possibilità di estensione. Acquisire le conoscenze per la stesura di uno scripting engine può essere dunque un buon investimento. In questo articolo vedremo come maneggiare un semplice linguaggio ispirato al Basic. L'approccio che seguiremo sarà simile a quello di Java: avremo un file sorgente, il compilatore che lo tradurrà in bytecode e la macchina virtuale per eseguire il codice prodotto (vedi Fig. 1). Nel CD allegato alla rivista e nel sito web *www.ioprogrammo.it,* è presente un'implementazione esemplificativa in C++ alla quale invitiamo a far riferimento durante la lettura.

LINGUAGGIO

Prima di ogni cosa pensiamo al linguaggio da realizzare, qualcosa che ci permetta di svolgere espressioni di calcolo ed assegnare il risultato ad una variabile che sarà possibile visualizzare tramite comando *printv.*

Es: *a = 1;*
b = (3+a)/2
printv b;

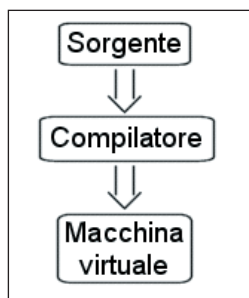
Fig. 1: ***Il file sorgente viene prima compilato per poi passare all'esecuzione della macchina virtuale.***

Per rendere più agevole la gestione dei calcoli introduciamo una semplificazione: opereremo con due soli elementi alla volta. Potremo sommare anche tre variabili ma dovremo fare uso di parentesi, altrimenti si commetterà errore. Tale limitazione sarà eliminabile in un secondo momento. Dunque, per a+b+c scriveremo a+(b+c). In questo modo (b+c) è trattato come se fosse un unico elemento, cioè il risultato dell'operazione che racchiude al suo interno. Il comando *input* permetterà all'utente di inserire, tramite tastiera, una cifra da memorizzare in una variabile, in questo modo si potranno creare programmi interattivi. Per concludere, *prints* per la visualizzazione di stringhe, ed i costrutti *if* e *while.* Questo linguaggio ci permetterà di scrivere qualcosa del tipo:

```
a=1; b=6; c=3;
while(a<b)
  if(a=c) then
    prints "siamo al 3!";
  else
    printv a;
  endif
  a = a+1;
wend
```

Questo semplice programmino conta i numeri da 1 a 5, ed avvisa l'utente quando arriva al 3. Ora che abbiamo inventato un linguaggio vediamo come utilizzarlo e passiamo al compilatore.

IL COMPILATORE

Un compilatore è un'applicazione capace di tradurre un programma da un linguaggio ad un altro. A noi interessa tradurre un file sorgente scritto nel linguaggio inventato, trasformandolo in bytecode. Questa operazione viene svolta attraverso due fasi: analisi e sintesi. Concentriamoci sulla prima. Il file sorgente si presenta al compilatore come una serie di caratteri. Deve prima distinguere le parole che lo interessano, identificando il loro tipo. Ad esempio gli servirà individuare le parole chiave (come *if, for, while*), le stringhe di testo, i valori numerici, etc. Questi simboli particolari vengono ricavati da un programma chiamato Lexical Analyzer (o più semplicemente Lexer), e sono detti token. Ogni linguaggio inoltre ha le sue regole, o meglio, la sua sintassi.

È necessario allora raggruppare questi token in strutture sintattiche che ci permettano di identificare i vari costrutti, secondo le regole che andremo a definire. Di quest'operazione se ne occupa un altro programma: il Parser. Fortunatamente, non dovremo scrivere noi questi programmi, lo farà qualcun altro in modo del tutto automatico e trasparente. Esistono infatti software che svolgono questa funzione: *Antlr* e *Dlg*. Scrivere Parser e Lexer efficienti è un'impresa difficile che si allaccia a concetti teorici avanzati ma, grazie a questi due programmi, potremo averne di ottimi a costo zero. È sufficiente fornire su file una descrizione del linguaggio da realizzare per ottenere direttamente i sorgenti C++ pronti per l'uso.

UN ASSAGGIO DI GRAMMATICA

Sorge spontanea una domanda: come si descrive un linguaggio? La lingua adoperata per farlo si chiama metalinguaggio. Ad esempio, un libro di C++ scritto in inglese adopera l'inglese come metalinguaggio. Tuttavia risulterebbe infruttuoso adottare una lingua parlata: quantomeno, si presenterebbero casi di ambiguità. Serve invece un modo rigoroso ed inequivocabile per descrivere le regole con esattezza matematica, così che possa capirle senza problemi anche *Antlr* e generare l'agognato parser. Per prima cosa dobbiamo trovare un modo per descrivere i token. *Antlr* permette di farlo tramite espressioni regolari. Un'espressione regolare è una descrizione che specifica in che modo una stringa può presentarsi.

Es: vogliamo identificare un numero in virgola. Sappiamo che esso inizia con uno o più caratteri compresi tra '0' e '9', seguiti da un '.' ed un'ulteriore serie di caratteri numerici. Ecco l'espressione regolare:

```
[0-9]+ { . [0-9]+ }
```

Il simbolo + ad apice sta ad indicare la presenza di almeno uno dei caratteri a scelta tra quelli compresi nell'intervallo da 0 a 9, le parentesi graffe servono invece a raggruppare la seconda parte dell'espressione rendendola opzionale (così includiamo anche i numeri senza virgola). In una grammatica Antlr questo si scrive:

```
#token FLOAT "[0-9]+ { . [0-9]+ }"
```

Il token FLOAT sarà dunque la versione testuale di un numero in virgola. Altro simbolo adoperato nelle espressioni regolari è *. Ad esempio *[a-z]** indica che possono presentarsi zero o più caratteri compresi tra *a-z*. Invitiamo il lettore a consultare il file *lang.g* nei sorgenti allegati per ulteriori esempi commentati.

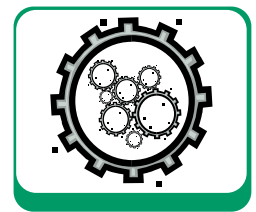
BACKUS-NAUR-FORM

La grammatica, che servirà ad individuare le strutture sintattiche tramite il Parser può essere adoperata anche per generare i vari costrutti. Quando elaboriamo una frase in italiano ci premuriamo di seguire lo schema "soggetto, predicato e complemento". Una cosa molto simile avviene quando scriviamo un *while* in C: immettiamo la parola chiave, facciamo seguire la condizione da verificare tra parentesi ed inseriamo poi il codice. Abbiamo dunque uno schema che ci indica come disporre i vari simboli. Chiameremo *regola* tale schema. Un linguaggio è, dunque, generato sulla via di un certo insieme di regole che ci indicano come comportarci mentre scriviamo un programma. Il Parser impiega questo schema, stavolta non per produrre nuovi costrutti ma per capire quali regole sono state adoperate in quelli presenti nel file sorgente. Come realizzare una simile meraviglia? Con la pubblicazione di *Algol 60 report*, nel 1960 Naur propose una notazione semplice ed efficace diffusasi con il nome di *Backus-Naur-Form*. Noi ci concentreremo sulla versione estesa supportata da Antlr, secondo la quale ogni regola segue la forma: *regola: elementi di sostituzione*. Ogni regola indica una sostituzione. Al lato sinistro prima dei ':' troviamo il nome della regola, al lato destro gli elementi che si possono mettere al suo posto, il tutto termina con un punto e virgola. Gli elementi di sostituzione possono essere combinazioni di token e nomi di altre regole che andranno successivamente definite. Vi possono essere diverse alternative di sostituzione che si specificano con il simbolo '|', es:

```
regola
: alternativa 1
| alternativa 2
...
| alternativa n
;
```

Qui la regola può essere sostituita da una delle alternative elencate, l'utilità è ovvia. Stabilito questo cominciamo con il descrivere il nostro linguaggio concentrandoci sulle espressioni, per i token fare riferimento al file *lang.g*

```
program : (statement)*;
statement: assignment;
assignment: IDENT EQOP assign ";";
assign: term {operate};
operate: (operazione term);
```

Immaginiamo che il nostro programma inizi con la regola *program*. Essa sarà sostituita da zero (nel caso di sorgente vuoto) o più *statement*. Per il momento lo *statement* rappresenta solo assegnazione *(assignment)* di risultati numerici in variabile *IDENT*. *EQOP* indica che la variabile alla quale destinare il

NOTA

È possibile specificare segmenti di codice c++ all'interno del file di grammatica. Ciò che si trova compreso tra i simboli << >> viene inserito nei sorgenti generati da Antlr.

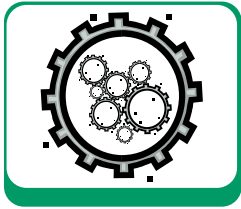
NOTA

In C++, è possibile trattare i token come oggetti per successive elaborazioni. Andando nel file di grammatica, basta specificare un nome seguito dal simbolo ':' prima del token interessato. Ad esempio:

```
rule: num:FLOAT;
```

Il token viene memorizzato nell'oggetto num ed è possibile accedere al testo in esso contenuto scrivendo

```
num->getText();
```

In genere restituisce un puntatore di tipo *char* *

risultato sarà seguita da un simbolo '=' ove procederà l'espressione di calcolo descritta dalla regola *assign*. Al posto di *operazione* troveremo alla fine il simbolo di somma, o di sottrazione, etc.

```
operazione
: somma
| sottrazione
| moltiplicazione
| divisione
;
term: identificatore | decimale | expr;
expr: LPAREN assign RPAREN;
```

Term rappresenta una variabile (*identificatore* sarà sostituita da *IDENT*), un numero decimale oppure un'ulteriore espressione. Questo permette di descrivere calcoli del tipo *a+((b+c)+d)* cioè con sotto-espressioni dentro. Infatti, *expr* viene sostituita dai simboli di parentesi e al loro interno è riproposta la regola assign, vista in precedenza. Questa specifica un ulteriore calcolo al cui interno è possibile trovare nuovamente sotto-espressioni. È importante comprendere il meccanismo. Ora la descrizione è completa, vediamo come adoperarla per generare il codice.

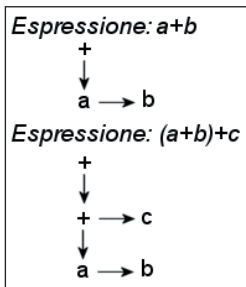



Fig. 2: Rappresentazione ad albero delle espressioni di calcolo.

ABSTRACT SYNTAX TREE

Il parser identifica le strutture sintattiche e ci dice in che ordine si presentano le regole. Possiamo dunque sapere quando si aprirà una nuova espressione o quando apparirà un'operazione di somma. Tuttavia non sarebbe saggio generare il codice con quello che abbiamo costruito sino ad ora. Prima conviene immagazzinare il tutto in una forma di rappresentazione intermedia, che si chiama Abstract Syntax Tree (AST). Una volta creato l'albero potremo generare il codice senza problemi. La generazione di AST è supportata da Antlr ma a noi interessa comprendere il meccanismo e quindi adoperiamo una nostra soluzione, più specifica per il nostro scopo. Nei sorgenti allegati un nodo AST è definito come segue:

```
class CASTBase
{ public:
  CASTBase();
  virtual int GetClassID();
  CASTBase *down, *right;
  std::string         Token;
protected:
  enum {kClassID = ID_CAST_BASE,};
};
```

NOTA

Esiste solo un'espressione regolare definita automaticamente, che riguarda la fine del file di input (End Of File). E' possibile rendere esplicita la definizione scrivendo: #token Eof "@". Il carattere @ infatti è usato per indicare il termine del file.

La funzione membro *GetClassID()* serve ad identificare la classe, restituendo un codice di riconoscimento. Tale metodo sarà impiegato anche nelle classi da essa derivate, come *CASTExpression* che, a parte una differente cifra di identificazione, non introduce nulla di nuovo. La stringa *Token* indica il simbolo associato. La rappresentazione che vogliamo realizzare è riportata in Fig. 2. Per esprimere la freccia a destra usiamo il puntatore *right,* per quello in basso *down.* Ci serve quindi un algoritmo che riesca a costruire l'albero adoperando come input l'espressione in forma testuale. Il parser, infatti, ci informa quando si sta aprendo una nuova sotto-espressione, quando si presenta un numero oppure il nome di una variabile. Possiamo scrivere queste informazioni in una stringa di testo per farla poi esaminare ad una procedura che la tradurrà in AST (l'espressione, passando dal parser risulterà corretta sintatticamente). Ricordiamo la semplificazione imposta, cioè che è possibile operare con due soli elementi alla volta, che siano numeri, variabili o sotto-espressioni. Quando troviamo un'espressione, dunque, sappiamo a priori che essa contiene al massimo due unità ed un simbolo di operazione inserito tra essi. Non ci interessa allora controllare quando il simbolo ')' chiuderà l'espressione poiché diamo per scontato che ciò avvenga sempre dopo il secondo elemento. L'espressione "*(a+(b+c))*" diviene quindi "*(a+(b+c*". Questo è sufficiente. Presentiamo ora una routine capace di tradurla in albero:

```
void SubParser(CASTBase *root)
{  /* mPtr punta all'espressione */
  if(*mPtr == ESPRESSIONE)
  { mPtr++;
    root->down = new CASTBase();
    SubParser(root->down);
    root->Token = *mPtr++;
    root->down->right = new CASTBase();
    SubParser(root->down->right); }
  else { root->ReadElement();}
}
```

Si tratta di una procedura ricorsiva. Quando trova il carattere di apertura parentesi, definito in ESPRESSIONE, capisce che il seguito del testo va trattato come sotto-espressione. Ricordiamo che una sotto-espressione è costituita da due elementi ed un simbolo di operazione in mezzo, la routine alloca un nuovo nodo al quale destinare il primo elemento ed invoca se stessa affinché quest'ultimo venga computato. Successivamente, viene prelevato il simbolo dell'operazione e memorizzato nella stringa Token del nodo attuale. Per passare alla seconda unità, nuovamente, la procedura alloca un nodo da destinarle ed invoca se stessa. Se invece la routine non incontra il simbolo di apertura parentesi conclude che l'elemento in questione, trattandosi di un numero o una variabile, non necessita di ulteriori elaborazioni ma solo essere letto e memorizzato nel

proprio nodo tramite *ReadElement()*. L'albero generato è quello della forma illustrata in Fig. 2.

PRIMA LE ISTRUZIONI

Prima di generare il codice finale dobbiamo farci una chiara idea di come questo debba essere. Definiamo quindi un insieme di istruzioni a basso livello che si occupino delle operazioni più elementari, in modo da produrre una sorta di linguaggio assembly fatto in casa. Nel file *compiler.h* troverete nell'enumerazione *InstructionSet,* preceduti da *INSTR_*, i nomi di tutte le istruzioni di cui abbiamo bisogno. Queste istruzioni vanno implementate in una macchina virtuale che sia capace di leggerle da file ed eseguirle una dopo l'altra. Anche di questo troverete i sorgenti C++ nel CD.

SOTTO CON IL CODICE!

A questo punto abbiamo tutti gli strumenti: disponiamo della struttura AST e sappiamo quali istruzioni adoperare. Non ci resta che generare il codice. Dobbiamo trovare un modo per tradurre l'albero nel sistema di istruzioni definito: sappiamo che ad ogni nodo è assegnato un token dal quale possiamo dedurne il significato. La prima cosa da fare dunque è controllarlo, esso può presentare un simbolo di operazione, un numero oppure il nome di una variabile. Nel primo caso, ricordando la fig. 2, sappiamo che i nodi in basso ed in basso a destra conterranno gli elementi da operare. Questi, però, possono essere a loro volta dei simboli di operazione. Adoperiamo, quindi, una procedura ricorsiva percorrendo ogni nodo della struttura AST secondo l'ordine delle operazioni da effettuare. Grazie a questo saremo in grado di generare il codice senza particolari problemi. Per il momento, immaginiamo che l'unica operazione effettuabile sia la somma e consideriamo soltanto il simbolo '+'. Scriveremo:

```
void CompileExpression(ASTBase *root)
{  char tok = *root->Token.c_str();
   switch(tok)
   { case '+':
        CompileExpression(root->down);
        CompileExpression(root->down->right);
        /* Ecco le istruzioni necessarie */
        *mOFile << INSTR_POP << endl;
        *mOFile << 'B' << endl;   // pop B
        *mOFile << INSTR_POP << endl;
        *mOFile << 'A' << endl;   // pop A
        *mOFile << INSTR_ADD << endl;
        *mOFile << "AB" << endl;   // add A,B
        *mOFile << INSTR_PUSH << endl;
        *mOFile << 'B' << endl;     // push B
```

```
        break;
     default:
        *mOFile << INSTR_MOVE << endl; // move Token,B
        *mOFile << root->Token << '-' << 'B' << endl;
        *mOFile << INSTR_PUSH << endl;
        *mOFile << 'B' << endl; // push B }
}
```

**mOfile* è il file stream per scrivere i dati su disco. La tecnica è questa: si copia nello stack ogni elemento che viene calcolato o prelevato dall'albero (nel caso si tratti di un numero o una variabile; in questo modo abbiamo sempre in esso gli ultimi elementi che

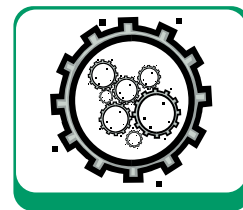
move op1, op2	Effettua una copia di op1 in op2
add op1, op2	Aggiunge op1 ad op2
sub op1, op2	Sottrae op1 a op2
mul op1, op2	Esegue op2=op2*op1
div op1,op2	Esegue op2=op2/op1
pop op	Preleva un float dallo stack e lo mette in op
push op	Carica la variabile op nello stack
gr op1,op2	Verifica se op1 è maggiore di op2
sm op1,op2	Verifica se op1 è minore di op2
eq op1,op2	Verifica se op1 è uguale a op2
jtp label	Se la verifica è positiva salta a label
jtn label	Se la verifica è negativa salta a label
jmp	Salta a label
prs text	Visualizza il testo ascii text
prv var	Visualizza il contenuto della variabile var
inp var	Chiede all'utente di immettere una cifra e la memorizza in var

TABELLA 1: Insieme delle istruzioni a basso livello.

sono stati operati. Se necessitiamo di ulteriori elaborazioni sappiamo dove andare a prendere i numeri che ci servono. Infatti, quando dobbiamo eseguire una somma possiamo direttamente prelevare due unità e sommarle tra loro, immettendo nuovamente il risultato nello stack. Come conseguenza, anche il risultato finale dell'espressione andrà a finire nello stack e quando dovremo assegnarlo ad una variabile le potremo tranquillamente farlo tramite l'istruzione *pop*. Un'occhiata al sorgente può chiarire ogni dubbio. Il codice, è ovvio, se venisse scritto a mano sarebbe più snello, ma qui ci interessa *comprendere* il meccanismo. L'ottimizzazione è un argomento molto complesso, specie per chi vuole iniziare. L'importante per ora è che funzioni.

CONCLUSIONI

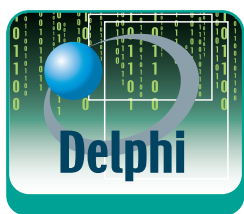
La programmazione di scripting engine è un argomento vasto e ricco di riferimenti a concetti teorici che sarebbe impossibile esaurire in poche pagine. Ad ogni modo si è cercata una via di esposizione libera di formalismi ed orientata alla pratica, sperando di attirare il lettore ad approfondire il tema e adoperare quanto appreso per i propri scopi. Se si presenterà il caso torneremo sull'argomento, per il momento vi auguriamo buona programmazione.

Andrea Ingegneri

Al sito www.antlr.org è disponibile la versione 2.0 di Antlr scritta in linguaggio java.

Array, record e funzioni ricorsive

Delphi: corso di Object Pascal

parte terza

Il parser di espressioni aritmetiche con cui stiamo apprendendo l'Object Pascal ha ancora un po' di lacune: parlando di routine, array, record ed altro ancora andremo a completarlo.

L'applicazione di questo articolo è stata creata con la seguente configurazione PC:

Pentium4 2.60GHz, 512Mb RAM

Sistema Operativo: Windows XP Home SP1

Software: Delphi Enterprise 7

Con l'obiettivo di studiare i fondamenti del linguaggio Delphi, siamo arrivati a costruire un interprete algebrico che riconosce ed esegue espressioni semplici con le quattro operazioni, oltre al modulo (resto di divisione), l'elevamento a potenza e la conversione tra basi diverse dalla decimale. Il nostro programma, però, non è in grado di gestire la priorità degli operatori aritmetici, né permette l'uso delle parentesi perché, con le conoscenze che avevamo nel numero scorso, sarebbe stato difficile implementare anche queste caratteristiche. Alla fine di questo articolo, invece, riusciremo ad avere finalmente un vero e proprio calcolatore esteso, con cui eseguire qualsiasi sequenza di calcolo con fino a tre (o più se volete) livelli di nidificazione, insieme alla comoda capacità di convertire da esadecimale, binario e ottale che già avevamo (vedo Fig. 1).

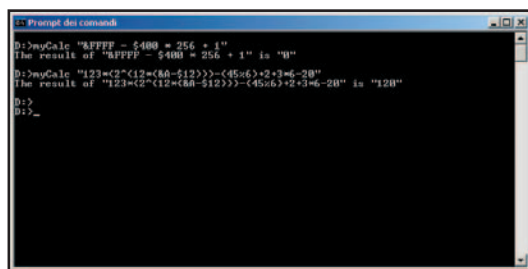



Fig. 1: Le possibilità dell'applicazione da console.

RIVEDIAMO LE ROUTINE

Per cominciare, separiamo la logica di parsing dal programma principale, creando una nuova unit (Parser) che offre due metodi, *ParseEspression* e *Tokenize.* Al fine di comprendere a fondo il codice su cui andremo a lavorare, dobbiamo solo aggiungere alcune informazioni alle nostre conoscenze sulle procedure e funzioni del Pascal. Già abbiamo visto che, nella dichiarazione delle routine, possiamo avere dei parametri che vengono passati al codice della routine stessa a suo uso e consumo. In Pascal i dati che tali parametri contengono sono passati per valore, ma esistono anche altre possibilità (vd. *Tabella 1*) che devono essere indicate esplicitamente al compilatore: vediamo quali sono! Innanzitutto, come potete immaginare, c'è il passaggio per riferimento, segnalato dalla parola chiave *var* anteposta all'identificatore.

Tipo	Keyword Pascal	Keyword C/C++
By Value	(default)	(default)
By Reference	var	&
Costante	const	Const
In Uscita	out	(non esiste)

TABELLA 1: I tipi di parametro del Pascal.

È poi previsto il passaggio di parametri costanti (come nel C o C++) a mezzo di *const,* sempre anteposto, ovvero, di valori che la routine garantisce di non variare nel corso della sua esecuzione (e il compilatore se ne assicura). Infine, Delphi prevede un utile tipologia che è quella dei parametri *out,* cioè che non contengono valori in entrata, ma che verranno inizializzati all'interno della routine ed il cui valore tornerà al chiamante. Quest'ultimo caso prevede che chi invoca la procedura o funzione predisponga una variabile il cui valore corrente sarà scartato per ricevere un valore di ritorno nella routine. È una operazione molto comune anche nel C++, sebbene non formalizzata sintatticamente e non assicurata dal compilatore, in codice del tipo

```
void function GetRect(CRect &rc);
CRect rect;
GetRect(rect);
```

Dove *rect* andrà a contenere il valore che *GetRect* imposterà. La differenza in Pascal è che un parametro *out* forza il fatto che il valore esterno della variabile *(rect)* non ha importanza annullandolo all'ingresso nella funzione. Questo non avviene in C++, dove stiamo ignorando il valore iniziale di un valore

passato semplicemente per riferimento. Vediamo adesso un esempio dei vari tipi di parametri che possiamo creare prima di tornare al codice dell'esempio:

```
procedure FunzioneEsempio(param1: string);
                              (* passaggio per valore *)
procedure FunzioneEsempio(var param1: string);
                         (* passaggio per riferimento *)
procedure FunzioneEsempio(const param: string);
                               (* parametro costante *)
procedure FunzioneEsempio(out param1: string);
                             (* parametro solo in uscita *)
```

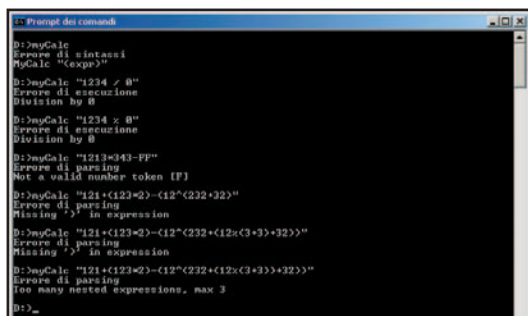



Fig. 2: ***I potenziali problemi dell'applicazione da console.***

Avrete riconosciuto nei metodi *ParseExpression* e *Tokenize* la presenza di un parametro in uscita *err*. L'utilità dei parametri in uscita è sovente quella di fornire un feedback sull'esecuzione del codice delle routine: il valore di ritorno di fatto è quello restituito eventualmente dalle funzioni, per cui l'informazione che viene data dai parametri *out* è normalmente accessoria. Nel nostro caso si tratta di un codice di errore a fronte del quale il chiamante potrà reagire in modo appropriato: questo consente tra l'altro di scrivere dei moduli che siano altamente portabili, in quanto determinano la condizione di errore, ma non la gestiscono, lasciandola a chi ha in mano il flusso principale dell'applicazione. I due metodi appena menzionati separano la logica di valutazione algebrica dal programma di console che interagisce con l'utente, per cui il compito di mostrare l'errore – lasciato appunto a chi usa l'unità – sarà gestito tramite un messaggio testuale sullo schermo, mentre un'applicazione grafica che utilizzi la stessa funzione potrebbe dedicare alla condizione di errore individuata tramite il parametro in uscita una message box apposita (vd. Fig. 2 e Fig. 3) Prima di procedere con l'applicazione d'esempio è ancora opportuno accennare alla possibilità di dichiarare delle funzioni overloaded in Pascal, così come avviene in Java e C/C++, anche se noi non ne faremo uso per ora. La differenza principale rispetto a questi altri linguaggi è che in Delphi è un errore creare due metodi con lo stesso nome anche se con parametri diversi senza che venga esplicitamente (qui sta la caratteristica del Pascal) indicato al com-

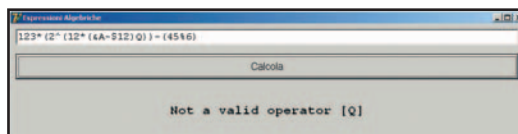



Fig. 3: ***Gestione dell'errore nell'applicazione grafica.***

pilatore che il metodo è overloaded: questa indicazione esplicita viene data ad opera della direttiva *overload* che segue la dichiarazione della routine, come di seguito mostrato

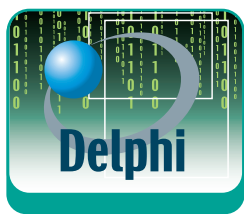
```
function Divide(X, Y: Real): Real; overload;
begin
  Result := X/Y;
end;
function Divide(X, Y: Integer): Integer; overload;
begin
  Result := X div Y;
end;
```

La stessa cosa in Java e C++ si può fare senza dare alcuna indicazione specifica al compilatore.
Tornando ora al parser matematico, per poter implementare la priorità degli operatori aritmetici abbiamo bisogno di un algoritmo un po' più complesso rispetto a quello illustrato il mese scorso: in particolare dobbiamo poter avere sott'occhio tutta l'espressione per poter determinare quali operazioni eseguire per prime. La procedura che utilizzerò non è probabilmente delle più efficienti, ma aiuta ad introdurre ed analizzare i concetti di programmazione che mi interessa illustrarvi in questo numero. Gli operatori sono stati suddivisi in quattro livelli di priorità, più uno che rappresenta i numeri, secondo lo schema della *Tabella 2*.

Valore	Operatore	Precedenza
4	^	Altissima
3	%	Alta
2	* /	Bassa
1	+ -	Bassissima
0	(numeri)	Nessuna

TABELLA 2: La precedenza degli operatori.

L'idea è la seguente: il metodo Tokenize prende la stringa ricevuta dalla riga di comando e la suddivide in token (numeri e operatori), memorizzandola in un array. Questo array viene poi analizzato una volta per ogni livello prioritario, a partire dal più alto fino al più basso, alla ricerca di operazioni da eseguire per quel livello, producendo ad ogni passaggio un risultando intermedio senza le operazioni appena eseguite. Così facendo si arriverà, dopo tutte le iterazioni, ad un risultato che rappresenta l'esecuzione algebrica dell'espressione iniziale. Tenterò di chiarire con un breve esempio: prendiamo l'espressione seguente

**È possibile scaricabile Delphi in versione di prova dal sito www.borland.com
È richiesta una registrazione gratuita a fronte della quale si riceverà la licenza temporanea di utilizzo del prodotto. Ai fini di questo corso, anche se non state testate, sono sicuramente adatte anche versioni precedenti di Delphi (5 o 6, per esempio).**

*2+4*4/2^3-7%3*

La tokenizzazione crea un array con i vari elementi nell'ordine corretto *(2,+,4,*,4,/,2,^,3,-,7,%,3)*, dopodichè il primo passaggio sarà al livello 4 e prevede l'esecuzione delle operazioni con ^, per cui i token risultanti dal primo passaggio corrisponderanno a questa espressione

*2+4*4/8-7%3*

Segue poi il livello 3 del % e quindi il risultato sarà

*2+4*4/8-1*

Il livello 2 prevede invece moltiplicazione e divisione, per cui

2+2-1

Infine l'ultimo livello eseguirà le addizioni e sottrazioni, ovverosia ci da il risultato finale che è *3*.
Come viene implementato tutto questo? Abbiamo due vuoti da colmare: il primo è quello degli array in Pascal, di cui non si è ancora discusso, ma che come potete immaginare non mancano. Il secondo è il riconoscimento della priorità dei simboli delle operazioni aritmetiche, che risolveremo creando una struttura di dati (chiamata record in Pascal) che memorizza per ogni token la sua priorità.

NOTA

IL VALORE SPECIALE NIL

Come *null* in Java e *NULL* in C/C++, la parola chiave *nil* in Pascal sta ad indicare che un riferimento non è valido. Nei tipi di riferimento tale valore speciale significa che la variabile che lo contiene non sta puntando a nessuna informazione valida e deve quindi ancora essere inizializzata. Per adesso noi abbiamo già visto due tipi di dato che accettano questo valore, ossia gli array dinamici e i tipi procedurali. Sull'argomento torneremo più in dettaglio quando parleremo di puntatori!

ANCORA SUI TIPI DI DATI

Per creare un array in Pascal abbiamo due sintassi che corrispondono rispettivamente agli array statici (con dimensione fissa) e a quelli dinamici (con dimensione variabile). Nel primo caso dovremo indicare l'indice del primo e dell'ultimo elemento dell'array in questo modo:

```
nomi: array[1..10] of string;
```

dove nomi sarà un insieme di 10 stringhe numerate da 1 a 10: notate che non siete vincolati ad iniziare da 1:

```
cognomi: array[11..20] of string;
```

è perfettamente valido! Per ottenere gli indici di riferimento di un array utilizzate le funzioni *Low()* e *High()* che vi restituiscono l'indice più basso e più alto rispettivamente, mentre *Length()* vi darà la dimensione dell'array.
Per quanto invece riguarda gli insiemi dinamici di elementi, la sintassi è molto simile, mancando solo la parte relativa agli indici:

```
punti: array of Integer;
```

Questi array sono creati senza dimensione, per cui è opportuno allocare spazio per essi con il comando *SetLength(array,n)* quando si devono inserire degli elementi: il metodo crea spazio per il numero di elementi indicato *n*, numerandoli da 0 a *n-1*. Lo stesso metodo può essere anche utilizzato per troncare e ridurre l'array ed i metodi che abbiamo menzionato prima per gli statici possono essere utilizzati anche in questo caso. Per referenziare un elemento di un array si utilizza la notazione con parentesi quadra, per cui dati gli esempi di prima potremmo accedere ai valori degli array creati in questo modo:

```
nomi[3] := 'federico';
cognomi[13] := 'mestrone';
SetLength(punti,4);
punti[2] := 30;
```

Un po' di attenzione è richiesta nell'utilizzo di array nelle dichiarazioni di funzioni e procedure. Non si può usare la parola chiave *array* come valore di ritorno di una funzione: è necessario creare un tipo di dati da poter utilizzare nella sintassi della stessa, in questo modo

```
type
Valori: array of string;
function GetValori(): Valori;
```

Invece nel caso del passaggio di array come parametro, la situazione è la seguente: gli array statici devono essere ridefiniti come tipo a parte (uguale all'esempio precedente)

```
type
SValori: array[1..15] of string;
procedure SetValoreStatico(staticparam: SValori);
```

così come per gli array dinamici

```
type
DValori: array of string;
procedure SetValoreDinamico(dynparam: DValori);
```

mentre un terzo tipo di parametro con questa forma

```
procedure SetValoreDinamico(openparam: array of string);
```

è detto "open array" ed indica che viene accettato sia un array statico che uno dinamico come valore per il parametro. Per questa casistica valgono però alcune restrizioni, tra cui l'impossibilità di passare *openparam* come parametro di *SetLength* e il fatto che l'indice più basso deve essere necessariamente 0. Ora, se con gli array possiamo gestire le liste di token delle nostre espressioni matematiche, con i

record possiamo creare degli elementi che mantengono insieme all'elemento dell'operazione anche il livello di priorità a cui è associato. I record sono come le *struct* del C, e consentono di creare tipi di dati che sono composti da diversi elementi. Nel caso dell'applicazione di esempio noi creiamo un tipo (Token) che è il seguente

```
type
  Token = record
    tk: string;
    p: Byte;
  end;
```

Ogni elemento di tipo Token consterà di una stringa che è il numero o l'operando e di un *Byte* che è il livello di priorità come da *Tabella 2*. Per accedere ai vari sottoelementi di tale valore composto si utilizza la classica notazione puntata nota a tutti i programmatori Java, VB, C, C++, ed altri ancora

```
var
myToken: Token;
begin
myToken.tk := '123';
myToken.p := 0;
end
```

Adesso, prima di poter andare a spulciare il codice di questa settimana, ci resta solo un nuovo tipo di dati, tipico del Pascal e dai risvolti molto interessanti. Partiamo da lontano: per rendere più mantenibile e modulare il codice del nostro esempio, ho creato delle funzioni in *CalcUtils* che si occupano delle varie operazioni aritmetiche e che ho chiamato *Sum, Subtract, Multiply, Divide, Modulus, Power*. Queste ci permettono di aggiungere controlli e funzionalità a queste operazioni senza mettere mano al codice di parsing (come per esempio nei metodi *Divide* e *Modulus*). A seconda del token di operazione tra due numeri invocherò uno dei metodi indicati. La soluzione più semplice è quella di utilizzare una struttura *case of* ed invocare la funzione giusta in base all'espressione. Il Pascal, però, offre una soluzione più elegante, quella cioè di utilizzare dei tipi procedurali, in altri termini delle variabili che contengono un riferimento ad una funzione e che possono essere usate come funzioni vere e proprie: la funzione che verrà effettivamente invocata dipende dal valore della variabile a run-time. Nel *Listato 1* trovate la definizione di *op*, che è un variabile procedurale che si riferisce ad una funzione che prende due interi e restituisce un intero. Più avanti nel codice trovare il seguente blocco:

```
begin
  case oldtks[cnt].tk[1] of
    '+': op := CalcUtils.Sum;
```

```
    '-': op := CalcUtils.Subtract;
    '*': op := CalcUtils.Multiply;
    '/': op := CalcUtils.Divide;
    '%': op := CalcUtils.Modulus;
    '^': op := CalcUtils.Power;
    else
      err := 'Not a valid operator [' + oldtks[cnt].tk + ']';
      Exit;
  end;
  Inc(cnt);
  tks[High(tks)].tk := IntToStr(op(StrToInt(
            tks[High(tks)].tk),StrToInt(oldtks[cnt].tk)));
end;
```

nella prima parte evidenziata assegniamo alla variabile op il riferimento ad una funzione matematica (uso la notazione puntata *CalcUtils.Sum* etc perché *Power* esiste anche nell'unità *Math*), mentre nella seconda parte evidenziata invochiamo la funzione che è stata referenziata tramite l'utilizzo di op come se fosse una funzione.

IL NUOVO PARSER

Riassumendo, il nuovo algoritmo procede in questo modo (lo trovate nel *Listato 1* presente sul CD o sul Web): la stringa viene tokenizzata in un array chiamato *tks* con *Tokenize,* mentre un secondo array di Token chiamato *oldtks* viene inizializzato con valore vuoto. Come si era detto prima, viene esaminato l'array di token. *tks* viene copiato in *oldtks* e svuotato, e si itera sugli elementi di quest'ultimo: se l'operazione in esame non è della priorità che si sta calcolando viene copiata nuovamente in *tks*, e si procede con l'elemento successivo. Questo viene ripetuto per tutti gli elementi e tutte le priorità, finché in *tks* resterà solo il risultato finale. Per quello che riguarda invece il gioco delle parentesi, ho sfruttato il fatto che il Pascal permette l'invocazione ricorsiva di routine, cioè la possibilità che durante l'esecuzione di una funzione venga fatta una chiamata alla stessa funzione in corso (vd Box 4). Avviene che durante l'elaborazione di *Tokenize* (che è parte dell'esecuzione di *ParseExpression*) non appena io trovo una parentesi tonda copio tutto quello che ci trovo dentro, comprese eventuali parentesi nidificate, in una stringa che passo tale e quale a *ParseExpression* nuovamente. In questo modo tutta l'espressione della parentesi viene calcolata ed inserita come token (numero) all'interno dell'espressione di livello più ampio.

Dal prossimo numero invece ci occuperemo di cose diverse, a partire dai puntatori per giungere poi alla programmazione ad oggetti, e cambieremo anche l'applicazione di fondo, per cui restate in allerta che ci sono novità in arrivo!

Federico Mestrone

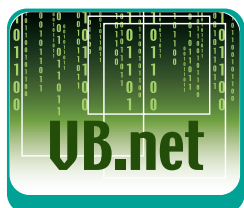
NOTA

FUNZIONI RICORSIVE

Le funzioni ricorsive sono delle routine che invocano se stesse. È importante notare che se non sono progettate con attenzione le funzioni ricorsive possono dare vita a dei loop infiniti da cui non si esce mai: sempre predisporre quindi una condizione di uscita dal ciclo! Inoltre tenete a mente che queste routine sono consumatrici di risorse perché ad ogni nidificazione di chiamata viene ad aumentare lo spazio occupato sullo stack di chiamata... per questo nel parser ho limitato a 3 il numero di parentesi nidificabili.

Il modello ADO.NET e le tecniche di accesso ai dati

L'accesso al Database

Inizia il viaggio all'interno degli strumenti disponibili in VB.Net per accedere ai dati di un database. Impareremo a connetterci e ad effettuare le prime interrogazioni.

La quasi totalità delle applicazioni VB.Net accede ai dati in varie forme, ma la maggior parte di questi dati proviene dai database. Lo scopo di questa serie di articoli sarà quello di descrivere gli strumenti di accesso ai dati, messi a disposizione da VB.Net, per recuperare ed aggiornare i dati in un database. La tecnologia che definisce il modello di programmazione ad oggetti per accedere ad una fonte dati, utilizzata in VB.Net si chiama ADO.NET. Utilizzando ADO .NET è possibile estrarre dati da Microsoft Access, Microsoft SQL Server o da sorgenti dati OLE DB più generiche, elaborarli ed aggiornare le tabelle originali del database. Descriveremo brevemente il generico modello ad oggetti di ADO .NET ed, in seguito, ci soffermeremo sull'impiego di ADO .NET per accedere ad un database Microsoft Access.

I PROVIDER DI DATI

I data provider di VB.NET hanno funzione di ponte tra l'applicazione e la sorgente dati. Descrivono un insieme di classi che consentono alle applicazioni di leggere e scrivere dati memorizzati in una fonte dati. Offrono, nel loro insieme un accesso veloce ed affidabile a specifiche origini dati (Sql Server, Access) oppure a qualsiasi fonte dati per la quale esista un driver ODBC. ADO.NET gestisce tre tipi di provider:

- **.NET OLE DB** che permette di accedere ad una sorgente dati per la quale esiste un provider OLE DB, ed è indicato per accedere a Microsoft Access basato sul motore Jet 4.0.
- **.NET SQL Server** ottimizzato per SQL Server 7.0, e SQL Server 2000.
- **.NET ODBC** che permette di accedere ad una sorgente dati per la quale esista un driver ODBC.

Per i nostri scopi, utilizzeremo il Data Provider .NET OLE DB.

IL MODELLO A OGGETTI DI ADO.NET

Qualunque sarà il Data Provider che vorrete utilizzare, si avranno sempre a disposizione quattro classi generiche:

Connection stabilisce una connessione ad un'origine dati specifica. La funzione dell'oggetto *Connection* è quella di stabilire una connessione alla fonte dati, permettendo di aprire e chiudere una connessione ad un database. Espone la proprietà *ConnectionString* che permette di definire, tra l'altro, il nome del database d'origine, i metodi *Open* e *Close* per aprire e chiudere la connessione ed il metodo *BeginTransaction* per avviare una transazione.
Command permette di eseguire un comando su un'origine dati. La funzione dell'oggetto *Command* è quella di consentire l'esecuzione di istruzioni SQL su un database. È possibile eseguire query d'interrogazione, inviare un qualsiasi comando sql (come un'istruzione di *insert* o di *update*), oppure invocare una stored procedure. Tutte queste operazioni sono possibili grazie ai vari metodi *Execute*.
DataReader permette di leggere un flusso di dati forward-only in sola lettura, proveniente da un'origine dati specifica. Il *DataReader* è utilizzato insieme all'oggetto *Command* e viene istanziato dopo una chiamata al metodo *ExecuteReader.* Mentre è aperto l'oggetto *DataReader* che utilizza un oggetto *Connection,* non è possibile utilizzare lo stesso oggetto *Connection* per nessun altro scopo se non chiudere la connessione.
DataAdapter consente la comunicazione tra una fonte dati ed un oggetto *DataSet* e risolve gli aggiornamenti con l'origine dati.

Il *DataSet* rappresenta l'oggetto principale dell'architettura disconnessa di ADO.NET. Il *DataSet* è paragonabile ad un database relazionale di piccole dimensioni memorizzato sul client e non dipende da alcun database specifico. Espone una collezione di oggetti *DataTable,* ognuno dei quali contiene un set di risultati tipicamente popolato da una query su una tabella del database. Un oggetto *DataTable* è costituito, a sua volta, da una collezione di oggetti *DataRow,* dove ciascuno di tali oggetti contiene un diverso record risultato dalla query di selezione. Il *DataSet* contiene, inoltre, una collezione di oggetti *DataRelation,* ognuno dei quali corrisponde ad una relazione tra oggetti

In VB.NET è ancora possibile utilizzare gli oggetti della precedente tecnologia ADO utilizzata per l'accesso ai dati in VB6. È sufficiente selezionare la voce di menu *Progetto–>Aggiungi* Riferimento e, nella finestra di dialogo, scegliere la componente *adodb* dalla scheda .NET.

DataTable differenti, in pratica come le relazioni che intercorrono tra le tabelle di un database relazionale. L'oggetto *DataAdapter* costituisce il ponte tra l'oggetto *Connection* e il *DataSet*. Con il suo metodo *Fill* si popola il *DataSet,* mentre con il metodo *Update* si aggiorna il database con i record modificati nel *DataSet*. Per utilizzare queste classi all'interno del codice, dovremo servirci del nome corrispondente nel Data Provider che andremo ad utilizzare, vediamoli in dettaglio

I NAMESPACE DI ADO.NET

Il Framework .NET è un'ampia raccolta di classi suddivise in un vasto insieme di namespace (spazi dei nomi), che raggruppano le classi simili. La maggior parte delle classi del Framework è riunita in un namespace denominato *System,* in particolare le classi cui fa riferimento ADO .NET sono raggruppate nei seguenti namespace:

System.Data contiene gli oggetti di ADO.NET che non fanno parte di uno specifico data provider. Ad esempio, questo namespace contiene l'oggetto *DataSet* e tutti i relativi oggetti secondari, come *DataTable, DataColumn, DataRow* e *DataRelation.*
System.Data.Common contiene le classi condivise dai provider di dati .NET come gli oggetti *DataAdapter* ed altre classi virtuali, utilizzate come classi di base per diversi oggetti appartenenti ai namespace seguenti.
System.Data.OleDb contiene le classi che costituiscono il provider di dati .NET Ole Db, come *OleDbConnection, OleDbCommand, OleDbDataReader* e *OleDbDataAdapter.*
Il namespace **System.Data.SqlClient** contiene le classi che costituiscono il provider di dati .NET SQL Server, come *SqlConnection, SqlCommand, SqlDataReader* e *SqlDataAdapter.*

In questa serie di articoli utilizzeremo i namespace *System.Data.OleDb* e *System.Data.* Proprio per la divisione delle classi in namespace, ogni volta che dovremo fare riferimento ad un oggetto specifico, il suo nome dovrà essere preceduto dal nome del namespace che lo contiene. Ad esempio per far riferimento ad un oggetto *OleDbConnection*, all'interno del codice, si dovrà scrivere:

```
System.Data.OleDb.OleDbConnection()
```

Per mantenere il codice il più compatto possibile, si può utilizzare l'istruzione *Imports* che semplifica l'accesso alle classi, eliminando la necessità di digitare in modo esplicito il nome completo del namespace che le contiene. Le istruzioni *Imports* devono sempre essere scritte nella parte superiore del file nel quale volete utilizzarle, prima di qualunque altro codice. Per queste ragioni, in tutti gli esempi di codice, assumeremo l'inserimento delle seguenti istruzioni *Imports:*

```
Imports System.Data
Imports System.Data.OleDb
```

OLEDBCONNECTION

L'oggetto *OleDbConnection* rappresenta una connessione attiva al database e necessita di alcune informazioni di base quali il nome e la posizione del database ed il driver da utilizzare per connettersi al database. La prima azione da eseguire quando si vuole utilizzare una fonte dati è quella di aprire una connessione verso quest'ultima, ciò significa creare un oggetto *OleDbConnection,* impostare la stringa di connessione ed aprire la connessione tramite il metodo *Open*. Riassumendo, si deve:

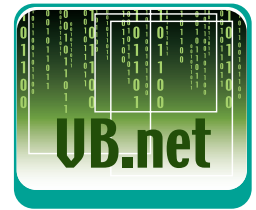
- Creare un oggetto *OleDbConnection.*
- Impostare la stringa di connessione ad uno specifico database.
- Aprire la connessione tramite il metodo *Open.*
- Eseguire le operazioni sul database, con uno degli oggetti Ado.Net che analizzeremo in seguito.
- Chiudere la connessione tramite il metodo *Close.*

Per creare un oggetto *OleDbConnection* si deve utilizzare la solita sintassi usata per creare un qualsiasi oggetto. Ad esempio possiamo scrivere:

```
Dim ObjConnection As New OleDbConnection()
```

CONNECTIONSTRING

La proprietà principale dell'oggetto *OleDbConnection* è *ConnectionString* che permette di impostare la stringa di connessione. La stringa di connessione consente di definire il nome del database d'origine ed altri parametri necessari a stabilire la connessione iniziale, delimitati da punto e virgola. Indichiamo di seguito alcuni dei possibili parametri da utilizzare:

Provider, che determina il nome del provider OLE DB da utilizzare per connettersi ai dati.
Ad esempio per connettersi ad Access: *Microsoft.Jet.OLEDB.4.0*
Data Source, che determina il nome del computer sul quale è installato il database, oppure il path completo di un database Access. È possibile usare come attributo localhost nel caso ci si connette ad SQL Server sulla macchina locale.
User ID che specifica lo username dell'utente che

ODBC è l'acronimo di Open Database Connectivity ed è il Protocollo standard di comunicazione, che permette il collegamento di applicazioni a vari server o file di database esterni.

esegue il login.
Password, che specifica la password dell'utente che esegue il login.
Initial Catalog, che specifica il nome del database da utilizzare.
Integrated Security specifica se utilizzare la protezione integrata per eseguire una connessione affidabile a SQL Server. Specificando il valore SSPI *(Security Support Provider Interface)* non è necessario utilizzare il nome utente e la password nella stringa di connessione, delegando al sistema operativo la gestione della sicurezza.

Supponendo di volersi collegare ad un database Microsoft Access dal nome *GestioneContatti* memorizzato in *C:\MieiDatabase,* si potrà quindi scrivere:

```
ObjConnection.ConnectionString =
                    "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\MieiDatabase\GestioneContatti.mdb;"
```

NOTA

In Ado.Net è possibile operare in *Modalità connessa* oppure in *Modalità disconnessa*. Nelle tradizionali applicazioni client/server, si stabilisce una connessione ad un database all'avvio dell'applicazione e si mantiene attiva durante tutto il ciclo di vita dell'applicazione. Questo approccio diminuisce il tempo di attesa quando si fanno continue operazioni sul database poiché elimina il tempo necessario alla connessione. In modalità disconnessa, invece, si apre una connessione, solo quando serve, si accede ai dati, e quindi si chiude la connessione per rilasciare le risorse lato server ad essa associate. Le applicazioni sono quindi connesse al database solo il tempo necessario per recuperare o aggiornare i dati.

APRIRE E CHIUDERE LA CONNESSIONE

Dopo aver istanziato un oggetto *OleDbConnection* con la opportuna stringa di connessione, è necessario richiamarne il metodo *Open* per collegarsi alla fonte dati. Possiamo ad esempio scrivere una generica procedura che si occupi di aprire la connessione:

```
Private Sub ApriConnessione()
    ObjConnection = New OleDbConnection()
    ObjConnection.ConnectionString =
                    "Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=C:\MieiDatabase\GestioneContatti.mdb;"
    ObjConnection.Open()
End Sub
```

Per liberare le risorse di un oggetto *OleDbConnection* quando non sono più necessarie, si deve richiamare il metodo *Close*. Possiamo ad esempio scrivere una procedura che si occupi di chiudere la connessione:

```
Private Sub ChiudiConnessione()
    ObjConnection.Close()
End Sub
```

Il metodo *Close* esegue il rollback di tutte le transazioni in sospeso e rilascia la connessione, se viene chiamato quando non ci sono connessioni aperte non viene generata alcuna eccezione. I metodi *Open* e *Close* non accettano argomenti. Quando si opera in modalità disconnessa diventa molto importante gestire gli errori, per evitare di lasciare connessioni inattive aperte, saturando il sistema. A causa del meccanismo di garbage collection di VB .NET, quando l'oggetto *OleDbConnection* esce dal proprio ambito di visibilità la connessione non viene chiusa automaticamente. Potrebbe essere chiusa diverso tempo dopo, per questo è importante chiudere esplicitamente l'oggetto *OleDbConnection*

OLEDBCOMMAND

Dopo aver aperto una connessione, siamo pronti per interagire con il database per mezzo dell'oggetto *OleDbCommand.* L'oggetto *OleDbCommand* deve essere associato ad un oggetto *OleDbConnection* preventivamente connesso alla sorgente dati, e può contenere una query di selezione (per leggere i dati dal database) o una query d'azione (per aggiornare i dati). Dopo aver impostato l'oggetto *OleDbCommand,* si esegue, quindi, uno dei relativi metodi *Execute:*

ExecuteNonQuery si utilizza per inviare la query d'azione specificata da *CommandText* al database, e restituisce il numero di record coinvolti. Permette attività di manipolazione di dati, come inserimenti, aggiornamenti e cancellazioni corrispondenti alle istruzioni SQL di *Insert, Update* e *Delete.*
ExecuteReader si utilizza per inviare la query di selezione specificata da *CommandText* al database, e restituisce l'oggetto *DataReader* che consente di accedere al set dei risultati *(resultset).*
ExecuteScalar si utilizza per inviare la query di selezione specificata da *CommandText* al database, e restituisce un singolo valore risultato di un'istruzione *Select* (valore scalare). Il metodo restituisce la prima colonna della prima riga di un gruppo di risultati, ignorando tutti gli altri valori. È consigliabile usare questo metodo, ad esempio, se il risultato è il frutto di query con clausole di aggregazione come *count* o *sum.*

L'oggetto *OleDbCommand* espone diverse proprietà, descriviamone alcune:

CommandText permette di impostare l'istruzione SQL della query o la stored procedure da eseguire sull'origine dati.
CommandType permette di impostare il valore che indica in che modo interpretare il tipo di query impostato nella proprietà *CommandText,* può assumere i valori: *Text, Stored-Procedure.*
Connection rappresenta l'oggetto *OleDbConnection* associato all'istanza corrente dell'oggetto *OleDbCommand.*
CommandTimeout permette di impostare il tempo di attesa (espresso in secondi) trascorso il quale l'esecuzione della query viene annullata e viene sollevata un'eccezione. Per default il suo valore è pari a 30 secondi.
Transaction rappresenta l'oggetto *Transaction* cor-

rispondente alla transazione in cui viene eseguito l'oggetto *OleDbCommand*.
Parameters contiene la collezione di tutti i parametri principali associati all'oggetto *OleDbCommand*.

IL DATABASE

Per gli esempi che seguiranno, si deve creare un nuovo database Microsoft Access dal nome *GestioneContatti*, in cui memorizzare le informazioni inerenti i nostri contatti con le aziende presso cui lavorano. Avremo quindi necessità di aprire Access e di creare due nuove tabelle: *Contatto, Azienda*. La tabella *Contatto* avrà i seguenti campi:

- *CodiceContatto*, campo chiave di tipo numerico
- *Nome*, di tipo testo
- *Cognome*, di tipo testo
- *Indirizzo*, di tipo testo
- *Comune*, di tipo testo
- *Telefono*, di tipo testo
- *Email*, di tipo testo
- *CodiceAzienda*, di tipo numerico

La tabella *Azienda* avrà i seguenti campi:

- *CodiceAzienda*, campo chiave di tipo numerico
- *RagioneSociale*, di tipo testo
- *Indirizzo*, di tipo testo
- *Comune*, di tipo testo

QUERY DI AZIONE SUL DATABASE

Le query di azione consentono di apportare variazioni sui dati contenuti nelle tabelle, operando contemporaneamente su più record e si dividono in:

Query di aggiornamento. Con questo tipo di query è possibile aggiornare dei valori di dati esistenti in una tabella.
Query di eliminazione. Consente di eliminare alcuni dati di una tabella.
Query di accodamento. Permette di inserire dei nuovi dati in una tabella.

Per eseguire una query di azione sul database, tipicamente si dovrà creare un oggetto *OleDbCommand* con le proprietà fondamentali *CommandText* e *OleDbConnection* ed eseguire il comando SQL tramite il metodo *ExecuteNonQuery*. Vediamo alcuni esempi.
Per inserire una nuova azienda nel nostro database, si deve:

- costruire la stringa corrispondente all'istruzione Sql di *Insert;*
- creare l'oggetto *OleDbCommand* passandogli la stringa sql;
- aprire la connessione;
- impostare la proprietà *Connection* all'oggetto *objConnection* definito in precedenza;
- eseguire il comando SQL tramite il metodo *ExecuteNonQuery;*
- Chiudere la connessione.

Per questo, utilizzando le procedure di apertura e chiusura della connessione, descritte in precedenza, si può scrivere:

```
Dim QuerySql As String = "INSERT INTO azienda
                         (CodiceAzienda, RagioneSociale,
Indirizzo, Comune) VALUES (1,'LuigiSoft','Via
                                        Verdi','Cosenza')"
Dim ComandoOleDb As New OleDbCommand(QuerySql)
ApriConnessione()
ComandoOleDb.Connection = ObjConnection
ComandoOleDb.ExecuteNonQuery()
ChiudiConnessione()
```

Analogamente per modificare la ragione sociale dell'azienda appena inserita possiamo seguire lo stesso schema, l'unica parte di codice che cambierà sarà, ovviamente, il comando SQL:

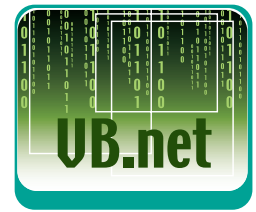
```
Dim QuerySql As String = "UPDATE azienda SET
                         RagioneSociale='SaraSoft' WHERE
CodiceAzienda=1"
Dim ComandoOleDb As New OleDbCommand(QuerySql)
ApriConnessione()
ComandoOleDb.Connection = ObjConnection
ComandoOleDb.ExecuteNonQuery()
ChiudiConnessione()
```

Infine per cancellare l'azienda possiamo utilizzare il comando *Delete* di SQL, e scrivere, sempre con lo stesso schema:

```
Dim QuerySql As String = "DELETE * FROM azienda
                                 WHERE CodiceAzienda=1"
Dim ComandoOleDb As New OleDbCommand(QuerySql)
ApriConnessione()
ComandoOleDb.Connection = ObjConnection
ComandoOleDb.ExecuteNonQuery()
ChiudiConnessione()
```

CONCLUSIONI

Con questo articolo abbiamo iniziato il viaggio all'interno della gestione dei database con VB.Net, occupandoci di descrivere gli strumenti necessari per collegarsi ad un database Microsoft Access e per eseguire su di esso delle query di azione.

Luigi Buono

La stringa di connessione può includere anche altri attributi:

• *Connection Timeout* imposta il tempo di attesa (espresso in secondi) trascorso il quale, se il tentativo di aprire la connessione fallisce viene sollevata un'eccezione (il default è di 15 secondi).

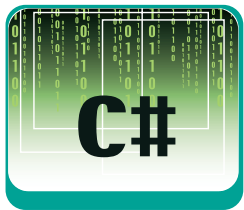
• *Packet Size* (solo per il Data Provider SQL Server) imposta le dimen-sioni dei pacchetti di rete utilizzati per comunicare con SQL Server. Può rive-larsi utile per ottimizzare il flusso di dati da e verso SQL Server.

• *Workstation ID* (solo per il Data Provider SQL Server) è una stringa che può essere utilizzata in un secondo momento per identificare il client.

Definire le proprie eccezioni

La gestione delle eccezioni

parte terza

Impareremo come realizzare eccezioni personalizzate, da usare all'interno delle nostre applicazioni e delle nostre librerie. Prenderemo così maggiore confidenza con la classe *System.Exception*.

Già diverse volte, nell'arco delle lezioni precedenti, ho decantato le lodi dei meccanismi per la gestione delle eccezioni incorporati in C# e Java. Probabilmente non mi stancherò mai di farlo. Come già ho detto, i moderni meccanismi di gestione delle eccezioni aiutano il programmatore nello sviluppo di software robusto, cioè poco vulnerabile alle situazioni inattese. La potenza della gestione delle eccezioni con .NET (e anche con Java), ad ogni modo, non si ferma qui: il meccanismo di base può essere espanso dal programmatore, in modo che anche le proprie applicazioni e le proprie librerie possano trarre vantaggio dall'utilizzo delle eccezioni. È sufficiente realizzare un proprio set di eccezioni, da lanciare e gestire quando lo si riterrà opportuno. In questa lezione scopriremo come fare.

DERIVARE LA CLASSE SYSTEM.EXCEPTION

La libreria di .NET contiene un buon numero di eccezioni, tutte collegate agli specifici argomenti che fanno parte del framework di .NET. Ad esempio, ci sono le eccezioni che riguardano la gestione dei flussi di input/output (*System.IO.FileNotFoundException, System.IO., System.IO.* e così via); ci sono quelle che riguardano il networking (*System. Net.Sockets*, per citarne una); ci sono le eccezioni relative ai database (*System. Data.*), e tante altre ancora.

Nonostante questo, è facile che un nuovo pacchetto o una nuova applicazione necessitino di uno specifico insieme di eccezioni progettate ad hoc. Per estendere e personalizzare il meccanismo di gestione delle eccezioni di C# e .NET, è sufficiente realizzare delle classi derivate, direttamente o indirettamente, da *System.Exception,* così come potete notare nel listato riportato di seguito:

```
class MyException : System.Exception {
  string dettagli;
  public MyException(string s) {
    dettagli = s;
}
  public void dimmiDettagli() {
    System.Console.WriteLine(dettagli); } }
```

MyExceptionestende System.Exception, pertanto è un'eccezione. Rispetto ad *Exception,* definisce il nuovo metodo *dimmiD().* Qualsiasi applicazione, ora, può fare uso della nuova eccezione, proprio come se fosse incorporata direttamente nella piattaforma .NET:

```
class Test {
  public static void Main() {
    try {
      throw new MyException("Eccezione per prova");
    } catch (MyException e) {
      e.dimmiDettagli(); }}
```

LA CLASSE SYSTEM.EXCEPTION

Tutte le classi di eccezione realizzate dal programmatore disporranno, per via delle norme dell'ereditarietà, dei metodi e delle proprietà di *System.Exception*. Pertanto, comprendere meglio i contenuti di questa classe può rivelarsi utile ed importante: spesso i programmatori, durante lo sviluppo delle proprie eccezioni, ridefiniscono o sfruttano i campi di *System.Exception*. *Exception* definisce numerose proprietà. Tra le più importanti vanno segnalate *Message*, *StackTrace* e *TargetSite*, che sono tutte stringhe di sola lettura. *Message* riporta un messaggio che descrive l'eccezione sollevata, *StackTrace* contiene l'elenco delle chiamate che ha attraversato l'eccezione, mentre *TargetSite* contiene il nome del

metodo che, per primo, ha scatenato l'eccezione. Un semplice test, basato sul classico esempio della divisione per zero, può dimostrarci l'utilità di queste tre proprietà:

```
class Test {
  public static void dividi(int a, int b) {
    System.Console.WriteLine(a + " / " + b + " = " + (a / b));
}
  public static void Main() {
    try {
      dividi(5, 0);
    } catch (System.Exception e) {
      System.Console.WriteLine("Messaggio: " +
                                          e.Message);
      System.Console.WriteLine("StackTrace: " +
                                          e.StackTrace);
      System.Console.WriteLine("TargetSite: " +
                                          e.TargetSite);} }
}
```

L'output di questo programma sarà:

Messaggio: *Tentativo di divisione per zero.*
StackTrace: *at Test.dividi(Int32 a, Int32 b)*
at Test.Main()
TargetSite: *Void dividi(Int32, Int32)*

Come è ora più facile osservare, *e.Message* riporta il messaggio collegato alla specifica eccezione ricevuta. Su *e.StackTrace* troviamo i metodi che l'eccezione e ha attraversato prima che noi la catturassimo. Il primo, difatti, è *dividi()*, mentre il secondo ed ultimo è *Main()*, all'interno del quale l'eccezione sta venendo gestita. Su *e.TargetSite* troviamo il punto di origine dell'eccezione: il metodo *dividi()*. Tutte queste informazioni, naturalmente, non sono pensate per essere mostrate all'utente. Lo scopo del programmatore è scrivere software robusto, che renda le eccezioni invisibili all'utente, gestendole e risolvendole una ad una. In fase di debug, ad ogni modo, queste tre informazioni sono fondamentali per capire da dove partono le eccezioni e quale è la loro tipologia. Oltre alle tre proprietà esaminate, *System.Exception* utilizza di frequente il proprio metodo *ToString()*, il cui scopo è fornire una rappresentazione in stringa dell'eccezione. Modifichiamo l'ultimo test eseguito, per verificare l'utilità del metodo in oggetto:

```
class Test {
  public static void dividi(int a, int b) {
    System.Console.WriteLine(a + " / " + b + " = " + (a / b));
  }
  public static void Main() {
    try {dividi(5, 0);
   }
   catch (System.Exception e) {
      System.Console.WriteLine(e.ToString());
   } }
}
```

L'output del programma è:

System.DivideByZeroException:
Tentativo di divisione per zero.
at Test.dividi(Int32 a, Int32 b)
at Test.Main()

Come è possibile osservare, il metodo *ToString()*, in questo caso, propone una serie di informazioni direttamente ricavate dalle tre proprietà esaminate in precedenza. Nell'esempio abbiamo gestito il generico tipo di eccezione *System.Exception*, ma quella che nello specifico è stata ottenuta è una *System.DivideByZeroException:* il metodo *ToString()* ce lo ha fatto scoprire. Infine, la classe *System.Exception* definisce diversi costruttori. Di nostro interesse sono i seguenti due:

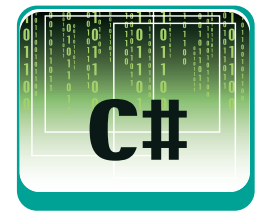
- **"Exception()** Costruttore privo di argomenti.
- **"Exception(string message)** Costruttore con un argomento, una stringa. Permette di creare un'eccezione alla quale sarà associato il messaggio espresso con l'argomento.

Ora che conosciamo meglio i contenuti della classe *System.Exception* possiamo avvantaggiarcene per sviluppare delle eccezioni personalizzate migliori.

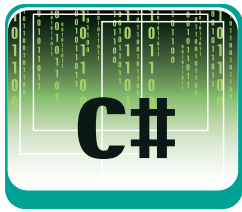
UN ESEMPIO

Supponiamo di dover realizzare un software che permetta di introdurre degli studenti in una aula in cui fare lezione. In maniera molto semplicistica, dovremo realizzare le classi *Studente* ed *Aula*. Cominciamo dalla prima:

```
class Studente {
  private string nome, cognome;
  public string Nome {
    get { return nome; } }
  public string Cognome {
    get { return cognome; } }
  public Studente(string n, string c) {
    nome = nome;
    cognome = cognome; } }
```

La classe *Studente* è molto semplice. Tutto quello che permette di fare è associare un nome ed un cognome allo studente, che si comporteranno come proprietà di sola lettura. La classe *Aula* dovrà avere una capienza massima e dovrà disporre di un metodo utile per inserire un nuovo studente all'interno dell'aula. Ma se la stanza è già piena (la capienza

• **GUIDA A C#**
Herbert Schildt
(MCGRAW-HILL)
ISBN 88-386-4264-8
2002

• **INTRODUZIONE A C#**
Eric Gunnerson
(Mondadori Informatica)
ISBN 88-8331-185-X
2001

• **C# GUIDA PER LO SVILUPPATORE**
***Simon Robinson* e altri**
(Hoepli)
ISBN 88-203-2962-X
2001

massima è stata raggiunta) non sarà possibile aggiungere nuovi studenti. La classe dovrà notificarlo al codice chiamante lanciando un'eccezione *AulaPienaException:*

```
class AulaPienaException : System.Exception {
  private int postiDisponibili;
  public int PostiDisponibili {
    get {return postiDisponibili;} }
  public AulaPienaException(string message, int pd) :
                                        base(message) {
     postiDisponibili = pd;}
  public override string ToString() {
    return base.ToString() + "\r\nPosti massimi
                            disponibili: " + postiDisponibili;}
}
```

AulaPienaException mette a disposizione del programmatore un solo costruttore, che accetta due argomenti. Il primo dei due è il messaggio di errore. Questo sarà passato al costruttore della classe base *System.Exception*, e quindi diverrà il messaggio di errore "ufficiale" dell'eccezione. Il secondo argomento deve riportare il numero di posti disponibili nell'aula. Il metodo *ToString()* è stato ridefinito. Nella stringa restituita, oltre alle informazioni già fornite dalla definizione della classe base, inserirà la capienza massima dell'aula che si sta cercando di riempire oltre la sua capacità. A questo punto, diventa possibile scrivere il codice della classe *Aula*:

```
class Aula {
  private Studente[] posti;
  private int prossimoPosto = 0;
  public Aula(int quantiPosti) {
    posti = new Studente[quantiPosti];}
  public void aggiungiStudente(Studente s) {
    if (prossimoPosto == posti.Length) {
      throw new AulaPienaException("L'aula è piena",
                                          posti.Length);
    } else {
      posti[prossimoPosto++] = s;} }
}
```

Ora potrete divertirvi a scrivere dei test sull'uso combinato delle classi *Studente* e *Aula*. Eccone uno:

```
class Test {
  public static void Main() {
    Aula aula = new Aula(3);
    aula.aggiungiStudente(new Studente("Mario", "Rossi"));
    aula.aggiungiStudente(new Studente("Luigi", "Bianchi"));
    aula.aggiungiStudente(new Studente("Antonio", "Verdi"));
    aula.aggiungiStudente(new Studente("Enzo", "Grigi")); }
}
```

In questo codice non c'è gestione delle eccezioni. Siccome si tenta di superare la capacità totale dell'aula rappresentata, l'esecuzione del programma stamperà in output:

Eccezione non gestita: AulaPienaException:
L'aula è piena
at Aula.aggiungiStudente(Studente s)
at Test.Main()
Posti massimi disponibili: 3

Un test più complesso ed interessante, che usa la gestione delle eccezioni per evitare messaggi di errore interni rivolti all'utente, è il seguente:

```
class Test {
  public static void Main() {
    int posti = 0;
    do {
      try {
        System.Console.Write("Quanti posti nell'aula? ");
        posti = int.Parse(System.Console.ReadLine());
        if (posti <= 0) System.Console.WriteLine(
                  "Introdurre un valore maggiore di zero.");
      } catch (System.FormatException e) {
        System.Console.WriteLine("Valore non valido..."); }
    } while (posti <= 0);
    Aula aula = new Aula(posti);
    while (true) {
      System.Console.Write("Vuoi aggiungere uno
                                    studente (s/n)? ");
      string s = System.Console.ReadLine();
      if (s == "n") break;
      else if (s == "s") {
        System.Console.Write("Nome: ");
        string nome = System.Console.ReadLine();
        System.Console.Write("Cognome: ");
        string cognome = System.Console.ReadLine();
        try {
          aula.aggiungiStudente(new Studente(nome,
                                             cognome));
          System.Console.WriteLine("Studente aggiunto!");
        } catch (AulaPienaException e) {
          System.Console.WriteLine("Impossibile
                                   aggiungere lo studente!");
          System.Console.WriteLine(
            "I " + e.PostiDisponibili + " posti disponibili
                                   sono esauriti." ); } }
    } }
}
```

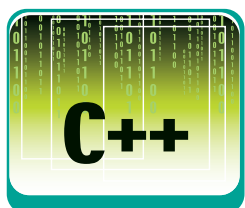
CONCLUSIONI

Con questa lezione termina la parte del corso dedicata alla gestione delle eccezioni. A partire dalla prossima lezione toccheremo nuovi argomenti. Come avremo modo di vedere, la gestione delle eccezioni ci tornerà spesso utile.

Carlo Pelliccia

Tecniche di ispezione e modifica dei contenitori

Algoritmi "Modifying" e "Non Modifying"

In questa lezione tratteremo gli algoritmi che permettono di estrapolare dati da un contenitore senza modificarlo e di quelli il cui scopo è, invece, proprio quello di "mettere mano" al contenitore per aggiornarlo.

La puntata scorsa abbiamo parlato degli algoritmi per effettuare il riordino (sorting) di un contenitore. Questa è solo una delle possibilità che ci vengono offerte dalla STL, in quanto possiamo pensare i nostri algoritmi divisi in classi che si occupano ognuna di implementare determinate funzionalità. A proposito della classificazione degli algoritmi, dobbiamo dire che quella vista la volta scorsa non è da considerarsi rigida, poiché un algoritmo può appartenere, in pratica, anche a più di una classe: nella classificazione quello che conta è la sua effettiva funzionalità. Ad esempio, l'algoritmo *sort()* potrebbe appartenere sia alla classe *"Modifying Sequence"*, sia alla classe *"Sorted Sequences"*, ma essendo la sua funzione primaria non la modifica ma l'ordinamento, allora tale algoritmo farà parte della classe *"Sorted Sequences"*.
Prima di procedere oltre, dobbiamo necessariamente dire che gli algoritmi presenti nella STL hanno sempre un nome autoesplicativo, seguito opzionalmente da due suffissi:

_if: con questo suffisso vengono indicati quegli algoritmi i quali fanno uso, oltre che dei parametri che verrebbero passati normalmente all'analogo algoritmo senza suffisso, di un function object (eventualmente definito da noi); ad esempio, l'algoritmo *find()* richiede come argomenti gli estremi della sequenza in cui cercare, e un terzo argomento che rappresenta il valore da cercare nella sequenza: se invece di un valore, si vuole cercare se c'è un elemento che verifichi una certa condizione, allora si usa l'algoritmo *find_if()*, passandogli come terzo argomento non più un valore ma un function object (facente le veci del predicato da verificare);

_copy: si indicano con questo suffisso quegli algoritmi che restituiscono un risultato copiato in una sequenza diversa da quella passata; ad esempio l'algoritmo *reverse()* inverte l'ordine di una certa sequenza, mentre *reverse_copy()* restituisce una nuova sequenza ottenuta da quella passata per argomento, invertendola.

È da notare che questi suffissi non vengono sempre utilizzati, e in effetti può capitare, come nel caso (visto la scorsa puntata) dell'algoritmo *sort()*, che dato un algoritmo l'analoga versione che fa uso di function object abbia lo stesso nome, ma un argomento diverso (un vero e proprio overloading dell'algoritmo).

NON-MODIFYING ALGORITHMS

Come anticipato la volta scorsa, in questa classe si trovano tutti quegli algoritmi che non modificano la sequenza su cui sono invocati (e nemmeno i singoli elementi della stessa). Gli algoritmi di questa classe possono essere invocati con tutti i contenitori, ma possono usare ovviamente solo iteratori in lettura, in particolare di tipo *input* o *forward*. Gli algoritmi di questa classe vengono utilizzati di solito per cercare all'interno di una sequenza, oppure per eseguire operazioni elementari quali contare elementi, estrapolare dati da specifici elementi, o confrontare elementi (o sequenze di elementi). Probabilmente uno degli algoritmi più utili è il *for_each()*, la cui firma è:

```
template<class In, class Op>
Op for_each(In first,In last,Op f)
```

Data la sequenza delimitata dagli iteratori *first* e *last* (entrambi di tipo *forward*), questo algoritmo chiama il function object *f* su ognuno degli elementi di tale sequenza, restituendo quindi al termine della sua esecuzione il function object *f* che è stato passato come terzo parametro.

Il *for_each()* viene classificato come *Non-Modifying*, ma in realtà esso può anche essere usato per modificare una sequenza, basta guardare il seguente esempio:

```
int N = numeroRandom();
vector<int> v(N);
numeroCasuale f;
for_each(v.begin(),v.end(),f);
```

In questo esempio si è supposto di avere il function object *numeroCasuale* che ci restituisce un numero a caso tra due estremi (potrete trovarne una versione banale nei codici allegati, ma sarebbe un utile esercizio tentare di farla da soli). Quanto eseguito dal nostro esempio è abbastanza semplice da capire e non richiede troppe spiegazioni: si noti che in questo caso (come voluto) il *for_each()* non lascia intatta la sequenza su cui è invocato (in questo caso, il vettore *v*), ma modifica i suoi elementi uno ad uno. È questo un esempio di algoritmo *Non-Modifying* di natura, che però può essere usato come un algoritmo di tipo *Modifying*. Alla classe *Non-Modifying* appartengono anche tutti quegli algoritmi che si occupano di effettuare ricerche all'interno di una sequenza. Tra di essi troviamo il *find()*, che ha la seguente firma:

```
template<class In, class C>
In find(In first,In last,const C& c)
```

Questo algoritmo restituisce un iteratore che punta alla prima occorrenza nella sequenza (delimitata dagli iteratori *first* e *last*) che assume un valore uguale a quello passato come terzo argomento. Ad esempio, tornando all'esempio precedente, potremmo scrivere:

```
vector<int>::iterator i = find(v.begin(),v.end(),9);
```

al posto dell'ultima istruzione, ed avremmo così un iteratore che punta alla prima occorrenza del numero 9 all'interno del nostro vettore di numeri casuali. Nel caso che all'interno della nostra sequenza non sia contenuta alcuna occorrenza del valore passato come terzo parametro, l'iteratore restituito è *last*. Di *find()* esiste, come abbiamo detto in precedenza, anche la versione che usa come terzo parametro un function object anziché un valore: il *find_if()*. Esso ha la seguente firma:

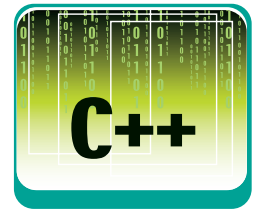
```
template<class In, class Op>
In find_if(In first,In last,Op p)
```

Questo algoritmo applica a tutti gli elementi della sequenza passata, uno dopo l'altro, l'operazione *p*: viene restituito anche qui un iteratore che si riferisce al primo elemento della sequenza per il quale l'operatore (passato come terzo argomento) assume il valore *true*.

Un'altra utile operazione che si può compiere senza modificare (in nessun modo) una sequenza passata è contare qualcosa. Ad esempio potremmo voler contare quante volte compare il numero 9 all'interno del nostro vettore di numeri casuali. Per far questo ci serve un altro algoritmo molto utile: il *count()*. Questo algoritmo restituisce il numero di occorrenze di un certo valore all'interno della sequenza passatagli, e ne esiste ovviamente una versione che invece di un valore verifica una condizione tramite l'uso di un function object: il *count_if()*.

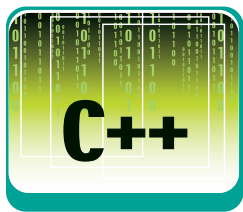
A questo punto, per risolvere il vitale problema di contare quanti 9 compaiono nel nostro vettore di numeri casuali, basterà usare una istruzione del tipo:

```
cout
  << "9 compare nella sequenza "
  << count(v.begin(),v.end(),9)
  << " volte!" << endl;
```

Un ultimo gruppo di algoritmi utili è quello che si occupa del controllo di uguaglianza tra sequenze. L'algoritmo principale a tal proposito è *equal()*, che riceve in ingresso tre parametri che sono rispettivamente: un iteratore che si riferisce al primo elemento della prima sequenza, uno che si riferisce al termine della sequenza, e un terzo che si riferisce all'inizio della seconda sequenza (da confrontare con la prima). Anche di questo algoritmo esiste la versione che controlla, invece della semplice uguaglianza, il rispetto di una condizione (che sarà di tipo binario, e sarà un *function object*) da parte delle due sequenze: questo algoritmo prende le due sequenze e verifica che le coppie di elementi corrispondenti all'interno di esse verifichino la condizione binaria passata come quarto argomento. Si noti che, come anticipato all'inizio, ci sono algoritmi i quali possono avere degli overload anzichè una versione identificata dal suffisso *"_if"*: è questo proprio il caso di *equal()*, la cui seconda versione (che fa uso dei *function*

APPROFONDIMENTI

A chi volesse approfondire la sua conoscenza sulle librerie standard del C++, consigliamo il validissimo libro THINKING IN C++ - 2ND ED. - VOLUME 2" *Bruce Eckel e Chuck Allison* che rappresenta sicuramente un ottimo riferimento per i programmatori più avanzati (o aspiranti tali) ed è oltretutto disponibile gratuitamente per il download, partendo dall'indirizzo http://www.mindview.net/Books/TICPP/ThinkingInCPP2e.html**.**

Se volete invece dare un'occhiata a una reference delle funzioni standard del C per la manipolazione di stringhe (o meglio: di array di caratteri terminati da "\0" :-) consultate l'indirizzo: http://www.cplusplus.com/ref/cstring/

Online è inoltre disponibile l'utilissimo "C++ Annotations" all'URL: http://www.icce.rug.nl/documents/ **che merita di essere visto (e letto) almeno una volta.**

object) non cambia il suo nome ed è in pratica un overload della versione più semplice: la differenza risiende nel fatto che la versione che usa un predicato binario per il confronto ha quattro argomenti invece di tre.
Complementare all'algoritmo *equal()* è l'algoritmo *mismatch()*, il quale restituisce una coppia di iteratori che punta alla prima coppia di elementi delle due sequenze che risultino diversi, oppure che non verifichino il predicato passato come quarto argomento. Il tipo restituito è un tipo contenitore pair (che rappresenta coppie di elementi), e la firma dell'algoritmo è:

```
template<class In, class In2>
pair<In,In2> mismatch(In first,In last,In2 first2)
```

che nella versione che usa un predicato diviene:

```
template<class In, class In2, class Op>
pair<In,In2> mismatch(In first,In last,In2 first2,Op p)
```

Una breve digressione su *pair* è d'obbligo, in quanto questo tipo è utilizzato in moltissimi contesti all'interno della STL. Spesso infatti è necessario estrapolare da un contenitore una coppia di valori, associati tra loro da una particolare relazione. Giusto per fare un esempio, una ricerca di un valore all'interno di un contenitore di tipo *map* è un *pair.* Il tipo *map* infatti contiene dei valori associati alle cosiddette "chiavi" *(keys)*, che altro non sono che un ulteriore valore (di solito un *int*) utilizzato per rendere univoco un oggetto all'interno del contenitore. Per chiarire la cosa potremmo pensare di costruire un oggetto di tipo *map* contente i nomi degli alunni di una scuola. Verosimilmente capiterà che vengano inseriti due nominativi uguali (cioè due studenti omonimi); ebbene sarà possibile distinguere il bravissimo "Stefano Rossi" della 4ª C dallo svogliato e ripetente "Stefano Rossi" della 3ª A semplicemente analizzando il valore della chiave che accompagna la stringa del nome all'interno del pair. Il concetto è analogo a quello di "chiave primaria" che forse sarà noto al lettore avvezzo alla programmazione di database: le chiavi sono tutte diverse tra loro, mentre i valori ad esse associati possono essere i medesimi.

MODIFYING ALGORITHMS

Gli algoritmi fin qui visti, permettono di effettuare una serie di operazioni su un contenitore, senza però modificarlo in alcun modo. Ovviamente la manipolazione con modifica è uno degli aspetti fondamentali dell'esistenza stessa dei contenitori, per cui deve poter essere effettuata in qualche maniera. Uno dei metodi possibili è quello di iterare "manualmente" all'interno del contenitore per effettuare i cambiamenti desiderati. Tuttavia ci sentiamo di sconsigliare questo modo di procedere, in favore di una metodologia più sistematica. L'utilizzo dei *"Modifying Algorithms"* è un ottimo metodo per risparmiare tempo&fatica (utilizzando funzioni già pronte) e scrivere codice chiaro e comprensibile. Gli algoritmi di questa categoria sono in numero davvero vasto e consentono di fare pressoché qualsiasi tipo di modifica sistematica su un contenitore. Uno dei più semplici e immediati è l'algoritmo *copy()* la cui firma è

```
template <class In, class Out>
Out copy (In first, In last, Out res);
```

copy() effettua (sorprendentemente!) la copia di un dato contenitore, che viene specificato tramite iteratori che puntano al primo elemento *(first)* e alla prima posizione dopo l'ultimo elemento *(last)*. Ovviamente è possibile copiare sotto-sequenze del contenitore specificando dei puntatori in posizioni interne (ad esempio per copiare gli elementi dal 5 al 15 in un nuovo vettore), cosa che rende omaggio alla grande flessibilità delle STL di cui più volte abbiamo parlato. Tale flessibilità si può riscontrare anche in una interessante caratteristica di *copy()* e cioè il fatto che il suo *output (res)* non deve necessariamente essere un contenitore; ad esempio il seguente codice è un modo elegante per stampare il contenuto di un vettore:

```
vector v;
//... v viene riempito qui ...
copy(v.begin(),v.end(),ostream_iterator<TIPO>(cout));
```

Quello che accade è che il contenuto di *v* viene copiato sullo stream di output *(cout)* per essere stampato a schermo. *copy()* presenta anche una variante che permette di effettuare in sicurezza copie di un contenitore il cui inizio della sequenza di output è all'interno della sequenza di input; la firma di questa copia all'indietro è:

```
template <class Bi, class Bi2>
Bi2 copy_backward (Bi first, Bi last, Bi2 res);
```

dove *Bi* sta a significare che abbiamo bisogno, in questo caso, di un iteratore di tipo *Bidirectional.* Ultima cosa da notare è che i suffissi *_if()* e *_copy()* non si applicano a questo algoritmo.
Se la cosa risulta comprensibile per quanto riguarda il significato astruso di un eventuale *copy_copy()*, l'assenza di un *copy_if()* che effettui la copia di un elemento se e solo se questo rispet-

ta un particolare vincolo, appare a tutti gli effetti una dimenticanza dei progettisti. Possiamo tuttavia risolvere facilmente questo problema, definendo la seguente funzione:

```
template <class In, class Out, class Pred>
Out copy_if (In first, In last, Out res, Pred p)
{
    while (first != last)
    {
        if (p(*first)) *res++ = *first;
        ++first;
    }
    return res;
}
```

Non è da escludere, comunque, che la particolare implementazione della STL che si sta usando includa già una funzione *copy_if()*.
Un altro algoritmo particolarmente utile è quello che consente di eliminare i duplicati adiacenti in una sequenza di elementi. Questo algoritmo è implementato nella funzione *unique()* utilizzabile nella sua forma base:

```
template <class For>
For unique(For first, For last);
```

oppure nella forma che accetta un predicato per il confronto di uguaglianza tra oggetti:

```
template <class For, class BinPred>
For unique(For first, For last, BinPred p);
```

dove le abbreviazioni *For* e *BinPred* stanno per *Forward Iterator* e *Binary Predicate*. L'utilizzo di *unique()* richiede una particolare attenzione in quanto l'eliminazione dei duplicati adiacenti (e solo di questi!) viene effettuata in una maniera particolare, diversa da quella che ci si potrebbe aspettare. L'algoritmo non fa altro che una scansione della sequenza di input, eliminando i duplicati, ma lasciando inalterata la sequenza che si trova dopo la sotto-sequenza iniziale di elementi non duplicati. Detto così sembra complesso, ma con un esempio risulterà banale.
Supponiamo di avere la sequenza:

AABBCCCD

la *unique()* applicata a questa sequenza restituirà una serie di caratteri composta, per le prime 4 posizioni, dai caratteri non duplicati, cioè:

ABCD

e nelle restanti 4 posizioni (la sequenza di input è lunga 8) copierà i valori originari. L'output sarà quindi dato da:

ABCDCCCD

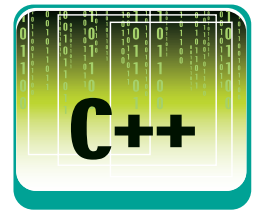
unique() quindi non elimina gli elementi ridondanti e, se vogliamo fare questa cosa, dobbiamo implementare a mano questa funzionalità. Per farlo è essenziale utilizzare il valore restituito da *unique()* che rappresenta il puntatore al primo elemento non facente parte della sequenza senza duplicati (nel nostro esempio la seconda 'C').
Il codice di questa funzione può essere ad esempio:

```
template <class C>
void EliminaDuplicati(C& c)
{
    sort(c.begin(),c.end()); //ordino il contenitore per
                                                rendere
            //"adiacenti" i suoi elementi duplicati
    C::iterator p = unique(c.begin(),c.end());
            // ho creato la sotto-sequenza di
            // elementi non duplicati
    c.erase(p,c.end());
            // cancellazione manuale degli
            // elementi superflui
}
```

unique() presenta anche la versione col suffisso *_copy()* che fornisce come output una copia della sequenza manipolata, lasciando inalterata l'originale.

CONCLUSIONI

In questa lezione abbiamo parlato delle due principali classi di algoritmi forniti dalla STL, i *Modifying* e i *Non Modifying*, e abbiamo visto alcune importanti e utili funzioni. Ci siamo soffermati con particolare attenzione alle relative firme delle varie funzioni, perché riteniamo importante la comprensione del comportamento di una funzione semplicemente dal suo nome e dalla lista dei parametri. Inoltre dare nomi autoesplicativi a una funzione è una delle abilità che un buon programmatore dovrebbe sviluppare. In ogni caso troverete un ampio esempio di codice per l'utilizzo degli algoritmi descritti sul CD-Rom allegato.
Vi invitiamo a dargli un'occhiata, magari effettuando delle piccole modifiche per verificare la comprensione esatta delle varie funzioni.
Nella prossima puntata concluderemo la panoramica sul (vasto) mondo degli algoritmi della STL parlando degli algoritmi *Sorted, Set, Min&Max* e *Heap:* non mancate!

Alfredo Marroccelli e Marco Del Gobbo

Se avete suggerimenti, critiche, dubbi o perplessità sugli argomenti trattati e vuoi proporle agli autori puoi scrivere agli indirizzi:

alfredo.marroccelli@ioprogrammo.it **e** marco.delgobbo@ioprogrammo.it

Questo contribuirà sicuramente a migliorare il lavoro di stesura delle prossime puntate.

Elementi di base: Costruttori e Stringhe

L'arte della costruzione

Ora che ti sei fatto le ossa con le basi della programmazione in Java sei pronto per affrontare un argomento importantissimo: i costruttori. Il menu del giorno è ricco e pieno di ingredienti, buona... lettura!

OBIETTIVI DI QUESTA LEZIONE

- Imparerai a passare dei parametri ai metodi.
- Spiegherò due o tre cose sulla classe *String*.
- Farai la conoscenza dei costruttori.

LA VECCHIA SCUOLA

In alcuni linguaggi "tradizionali" come il C, gli equivalenti dei metodi di Java possono avere due diversi nomi. Quelli che restituiscono un valore sono chiamati *funzioni* e quelli che non restituiscono alcun valore sono chiamati *procedure*. In Java non si fanno distinzioni esplicite tra i due casi, e si usa sempre il nome "metodi".

Finora hai incontrato solo due tipi di metodi: quelli che restituiscono un valore, e quelli che non restituiscono niente. Ecco una classe che contiene metodi di entrambi i tipi:

```
class Contatore {
  private int valore = 0;
  void incrementa() {
    valore++;
  }
  void decrementa() {
    valore--;
  }
  void stampa() {
    System.out.println(valore);
  }
  boolean positivo() {
    return valore > 0;
  }
}
```

Il *Contatore* ha un campo di nome *valore*. Il campo è privato, quindi non accessibile dall'esterno. Se crei un *Contatore,* il suo valore sarà inizialmente uguale a zero. Il metodo *void incrementa()* incrementa di uno il valore, e il metodo *void decrementa()* lo decrementa di uno (l'operatore --, che finora non avevi mai incontrato, è l'opposto dell'operatore ++). Un terzo metodo *void,* di nome *stampa(),* si occupa di stampare il *Contatore.* Il metodo *positivo()* appartiene invece alla categoria dei metodi che restituiscono un valore. La parola chiave *return* è obbligatoria nei metodi di questo genere. In particolare, *positivo()* restituisce *true* se e solo se il valore attuale del *Contatore* è maggiore di zero.
Fin qui non c'è niente di nuovo. Ma i metodi hanno un'altra possibilità: quella di ricevere dei valori dall'esterno. Ad esempio possiamo aggiungiamo un metodo al *Contatore* per verificare se il suo valore è superiore a quello di un altro *Contatore*:

```
class Contatore...
  boolean superioreA(Contatore altroContatore) {
    return valore > altroContatore.valore;
  }
```

Le parentesi tonde di questo metodo non sono vuote come quelle degli altri metodi che hai incontrato finora, ma contengono una *lista di argomenti* (in questo caso la lista è costituita da un solo argomento). Questo argomento è dichiarato nello stesso modo in cui si dichiara un qualsiasi oggetto: prima il suo tipo, poi il suo nome. Il tipo può essere un oggetto o un *tipo primitivo* (cioè qualsiasi cosa che non sia un oggetto, come ad esempio *int* o *double*). In questo caso, quello di cui ha bisogno il metodo è proprio un oggetto, e per la precisione un altro *Contatore.* Quindi il metodo *superioreA()* della classe *Contatore* prende un oggetto di tipo *Contatore* e restituisce un valore di tipo *boolean* (nota che anche il tipo di ritorno, proprio come il tipo dell'argomento, potrebbe tranquillamente essere un oggetto). Per la precisione, il metodo restituisce *true* se e solo se il valore di questo *Contatore* è superiore al valore del *Contatore* passato come parametro. Quindi nell'operazione sono coinvolti due contatori: uno "attivo" che riceve la chiamata del metodo e uno "passivo" che viene passato come parametro. Per fare riferimento al campo *valore* del primo contatore ti basta scrivere il nome del campo, perché siamo all'interno dello stesso oggetto. Per fare riferimento al *valore* dell'altro contatore devi invece scrivere il nome del parametro seguito da un punto e dal nome del campo. Nota anche che puoi accedere al campo *valore* anche se lo hai dichiarato *private*: la parola chiave private rende sì "invisibile" il campo, ma solo al di fuori di questa classe. In altre parole, un oggetto di classe *X* può sempre accedere a tutti i campi privati degli altri oggetti di classe *X*, che sono invece invisibili al codice contenuto in qualsiasi altra classe. Se sei confuso, prova a far girare questo programma:

```
class ProvaContatori {
  public static void main(String[] args) {
    Contatore primoContatore = new Contatore();
                                        // il contatore 1 vale 0
    Contatore secondoContatore = new Contatore();
                                        // il contatore 2 vale 0
    System.out.println(primoContatore.superioreA(
                           secondoContatore)); // false
    System.out.println(secondoContatore.superioreA(
```

```
                                primoContatore)); // false
    primoContatore.incrementa();  // il primo contatore
                                                   vale 1
    System.out.println(primoContatore.superioreA(
                            secondoContatore)); // true
  }
}
```

La prima stampa ha come risultato *false*, la seconda *true*. Quando chiami il metodo *superioreA()* devi passargli tra parentesi tonde un parametro di tipo *Contatore*. Se te ne dimentichi il compilatore ti fermerà con un messaggio di errore simile a questo:

```
superioreA(Contatore) in Contatore cannot be applied to ()
```

Cioè: il metodo *superioreA()* della classe *Contatore* non può essere usato con una lista di argomenti vuota. Un errore simile salta fuori anche se passi come parametro qualsiasi cosa che non sia un *Contatore*. Finalmente sai a cosa servono le parentesi tonde che seguono tutte le dichiarazioni e le chiamate ai metodi: contengono la lista degli argomenti, anche se questa lista può essere vuota.
Detto questo, cerchiamo di capire meglio cosa succede quando chiami il metodo.

CON QUALUNQUE ALTRO NOME

Osserva più da vicino la prima chiamata al metodo *superioreA()* nel programma *ProvaContatori*:

```
primoContatore.superioreA(secondoContatore)
```

Questa istruzione chiama il metodo *superioreA()* dell'oggetto *primoContatore*, passando come parametro l'oggetto *secondoContatore*. A questo punto il sistema "entra nel metodo". All'interno del metodo il parametro ha un nome diverso: *altroContatore*.
È come se all'oggetto *secondoContatore* venisse applicata un'etichetta con il nome *altroContatore*. Questa etichetta vale solo all'interno del metodo, e non è più visibile quando il metodo termina.
Grazie a questo meccanismo di "etichettamento", la riga successiva fa la stessa operazione invertendo i due contatori:

```
secondoContatore.superioreA(primoContatore)
```

In questo caso è il *secondoContatore* che viene chiamato, mentre il primo viene passato come parametro. Dato che i due contatori hanno lo stesso valore, il confronto tra i due campi *valore* non è mai verificato ed entrambe le chiamate restituiscono *false*.
Tutte le chiamate al metodo *superioreA()* restituiscono un valore booleano. In tutti e tre i casi, questo valore viene immediatamente passato come parametro all'operazione di stampa:

```
System.out.println(primoContatore.superioreA(secondo
                                        Contatore));
```

Questa istruzione stampa il valore restituito dal metodo, esattamente come se fosse una costante o il risultato di una semplice espressione. In futuro scoprirai che anche *System.out.println()* è un metodo, per quanto un po' particolare. Questo meccanismo, per cui il valore restituito da un metodo viene immediatamente passato come parametro ad un altro metodo, è molto comune nei programmi Java. L'alternativa, più macchinosa e alla lunga meno leggibile, è quella di conservare il valore in una variabile temporanea:

```
boolean b = primoContatore.superioreA(
                                secondoContatore);
System.out.println(b);
```

Se usi una variabile temporanea puoi conservare il valore restituito dal metodo anche dopo averlo stampato. Ma se il valore di *b* non ti serve più dopo l'operazione di stampa, allora è meglio passarlo direttamente a *System.out.println()* e poi lasciare che vada perduto.
Puoi avere anche metodi che prendono più argomenti. Ti basta separarli con delle virgole:

```
int sommaEProdotto(int a, int b, int c) {
  return (a + b) * c;
}
```

Questo metodo non fa riferimento a campi interni, quindi può appartenere a qualsiasi classe. Se ad esempio chiami: *sommaEProdotto(3, num, 5)*, dove la variabile *num* vale *4*, allora il risultato sarà *35* (in questo caso ho usato insieme variabili e valori numerici costanti). Naturalmente i parametri vengono letti ed "etichettati" dal metodo nello stesso ordine in cui li passi.
Ora puoi farti un'idea più generale di come è fatto un metodo: è una specie di "scatola" con una serie di tubi in ingresso e un unico tubo in uscita. I tubi in ingresso, che possono essere in un numero qualsiasi, prendono dei parametri (un metodo può anche non avere alcun tubo in ingresso, nel qual caso le sue parentesi tonde sono vuote). Il codice contenuto nel metodo può usare questi parametri come gli pare. Alla fine, l'eventuale risultato viene restituito dal tubo in uscita (che può anche non esistere, e in questo caso il metodo deve essere dichiarato *void*).
Ora ne sai abbastanza per affrontare un argomento importantissimo nella programmazione a oggetti: i costruttori. Ma prima facciamo una breve digressione a proposito delle nostre vecchie amiche stringhe.

Osserva questa riga di codice, che usa un oggetto di nome *c* e di classe *Contatore*:

```
boolean b =
        c.superioreA(c);
```

Qualunque sia il valore del *Contatore c*, il valore di *b* sarà sempre lo stesso. Quale? Se non sei sicuro della risposta, prova ad aggiungere questa riga di codice in fondo al programma *ProvaContatori* e a stampare il valore di *b*.

FILOSOFIA DEI METODI

Il nome di un metodo dovrebbe spiegare chiaramente cosa fa il metodo. Anche i nomi dei parametri devono essere chiari. Se non riesci a dare un buon nome al tuo metodo, allora probabilmente il tuo metodo fa "troppe cose". Un metodo ideale fa una cosa sola: ad esempio, i metodi che restituiscono un valore non dovrebbero fare nient'altro che quello. Un metodo che calcola un valore, lo stampa e incrementa un campo dell'oggetto dovrebbe essere spezzato in due o tre metodi diversi per le varie operazioni.

Aggiungi alla classe *Contatore* un metodo superioreAEntrambi()che prende altri due contatori, e restituisce true se e solo se il Contatore sul quale chiami il metodo ha un valore superiore a quelli di entrambi i Contatori che passi come parametri.

BELLO CARICO

L'operatore + è l'unico in tutto il linguaggio Java che ha due diversi significati a seconda del tipo dei suoi operandi: significa "concatenamento" se il primo operando è una *String*, "somma" se entrambi gli operandi sono int. Per questo motivo si dice che + è un'operatore overloaded (letteralmente "sovraccarico"), nel senso che ha più significati anziché uno solo.

Aggiungi alla classe *Contatore* un metodo *incrementaDi()* che prende un intero e incrementa il *Contatore* del valore intero passato. Scrivi un programmino per verificare che il metodo funzioni.

MONDO STRINGA

Finora hai incontrato le "stringhe", (cioè sequenza di caratteri), solo nelle istruzioni di stampa:

```
System.out.println("Questa è una stringa");
```

Questa stringa ha un valore costante, determinato dai caratteri tra virgolette. Puoi usare le stringhe anche come variabili. In Java esiste una classe *String* che serve proprio a trattare oggetti di tipo stringa. Non hai bisogno di dichiarare la classe: è già disponibile nel linguaggio, e puoi usarla quando vuoi.
Ad esempio:

```
String s = "Questa è un'altra stringa";
System.out.println(s);
```

Le stringhe non sono variabili primitive, ma oggetti (leggi le barre laterali se non conosci la differenza tra oggetti e variabili primitive). Il tipo *String* è quindi una classe "predefinita" da Java. Si tratta però di una classe particolare, per almeno due motivi. Il primo motivo è che per creare una *String* non devi usare la parola chiave *new*, che è invece sempre obbligatoria quando vuoi creare oggetti di qualsiasi altra classe. Per creare una stringa, invece, ti basta dichiararne il nome e assegnarle una costante:

```
String s = "abc";
```

Una seconda differenza tra i normali oggetti e le stringhe è nel fatto che queste ultime dispongono di un operatore tutto per loro: l'operatore di concatenamento +, che hai già incontrato nei mesi scorsi. Ad esempio:

```
System.out.println(s + "xyz" + s);  // stampa "abcxyzabc"
```

Tutti gli altri operatori di Java funzionano solo su variabili primitive. L'operatore di concatenazione è l'unico che funziona su oggetti. I creatori di Java avrebbero potuto ottenere lo stesso risultato aggiungendo alla classe *String* un metodo *concatena()* che prendesse come parametro un'altra *String*, ma in questo caso sarebbe stato scomodo e confuso concatenare più stringhe nell'ambito della stessa istruzione.

CIASCUNO A SUO MODO

Gli oggetti che hai creato finora sono "fatti con lo stampino". Alla creazione, tutti gli oggetti sono uguali. Dopo averli creati, li puoi modificare assegnando dei valori ai loro campi (quando questi campi non sono privati) o chiamando i loro metodi. Ad esempio, tutti gli oggetti di tipo *Contatore* nascono con un *valore* uguale a 0.
Questo modo di procedere può diventare piuttosto innaturale. Immagina ad esempio di voler costruire una classe *Persona* per un sistema anagrafico. Un oggetto di questa classe contiene i dati di una persona: nome, cognome, codice fiscale… Anzi, per semplificare gli esempi mi limiterò al nome e al cognome.
Una soluzione potrebbe essere quella di creare delle *Persone* con un nome e un cognome di default (ad esempio una stringa vuota, "") e un metodo che permette di impostarne il valore:

```
class Persona {
  private String _nome = "";
  private String _cognome = "";
  void assegnaNomeECognome(String nome, String
                                   cognome) {
    _nome = nome;
    _cognome = cognome;
  }
}
```

Ho seguito una convenzione che a me non dispiace, cioè quella di mettere come prefisso ai nomi di tutti i campi privati un carattere di *underscore* ("sottolineato"). Questa convenzione elimina eventuali ambiguità tra il nome dei campi e quello dei parametri nel metodo *assegnaNomeECognome()*, che serve appunto ad assegnare nome e cognome ad una *Persona*. Avrei anche potuto scrivere due metodi separati per assegnare il nome e il cognome, ma ho dato per scontato che i due valori vengano sempre assegnati contemporaneamente. Nota anche che i due campi sono privati, quindi ci servirebbe anche un metodo per recuperarne (o almeno stamparne) il valore - altrimenti non potremo mai sapere come si chiama una determinata *Persona*.
Ora possiamo creare una persona e assegnarle nome e cognome:

```
Persona p = new Persona();
p.assegnaNomeECognome("Paolo", "Perrotta");
```

Non particolarmente elegante, vero? Il problema è che queste due operazioni vanno sempre fatte in sequenza. Non ha senso creare una persona e poi non assegnarle un nome e un cognome. C'è anche da considerare il fatto che il nome e il cognome dovrebbero essere impostati una volta sola, mentre questa classe permette al client di modificarli tutte le volte che vuole, il che non è particolarmente sicuro. Insomma, sarebbe bello avere un meccanismo per assegnare il nome e il cognome a ciascuna *Persona* una ed una sola volta, nello stesso momento in cui la *Persona* nasce. Per fortuna questo meccanismo esiste, nella forma di una specie di metodo molto par-

ticolare chiamato *costruttore*, che viene chiamato automaticamente quando crei un oggetti con la parola chiave *new*. Ho detto “una specie”, perché il costruttore non è propriamente un metodo. Però ci somiglia: proprio come i metodi, ha un nome e una lista degli argomenti. Solo che il suo nome deve essere sempre identico al nome della classe, incluse le lettere maiuscole e minuscole. Grazie a questa omonimia, Java distingue subito i costruttori dai normali metodi. Inoltre, il costruttore non ha alcun valore di ritorno – nemmeno *void*. Ad esempio, una nuova versione della classe *Persona* potrebbe essere:

```
class Persona {
  private String _nome;
  private String _cognome;
  Persona(String nome, String cognome) {
    // inizializza i campi privati
    // con il valore dei parametri
    _nome = nome;
    _cognome = cognome;
  }
}
```

Ora non puoi più creare una persona come avresti fatto prima:

```
Persona p = new Persona();  // errore!
```

Se ci provi, il compilatore si arrabbia. Devi invece passare al costruttore i suoi due argomenti di tipo *String*:

```
Persona p = new Persona(“Paolo”, “Perrotta”);
```

Il costruttore viene chiamato sempre automaticamente, e solo in occasione della creazione di un oggetto. Non è possibile chiamarlo su un oggetto già esistente. Il costruttore di *Persona* assegna un valore ai due campi privati della classe durante la costruzione, il che ci garantisce due cose: che i due campi avranno un valore, e che questo valore non potrà più essere cambiato (a meno che naturalmente tu non scriva qualche altro metodo della classe *Persona* che cambia il valore dei campi). Allora, come mai fino ad ora hai tranquillamente creato degli oggetti anche se non avevi mai definito un costruttore nelle tua classi? La risposta è nascosta nel funzionamento del compilatore Java. Ogni volta che compili una classe senza costruttore, il compilatore se ne accorge e “scrive” automaticamente per te un *costruttore di default*. Il costruttore di default è vuoto, e non ha argomenti. Ad esempio, quando compili la classe *Contatore* il compilatore Java le aggiunge automaticamente un costruttore simile a questo:

```
class Contatore {
 Contatore() {}
```

```
 [...]
}
```

Tutto questo spiega anche il perché delle due parentesi tonde obbligatorie che seguono sempre il nome della classe quando usi la parola chiave *new*. Anche se non lo sapevi, hai sempre chiamato il costruttore di default, che non ha argomenti – e proprio come un metodo senza argomenti, richiede due belle parentesi vuote.

PER OGGI BASTA COSÌ

Concludiamo l’articolo di questo mese con una curiosità sulle stringhe. Poco fa ti ho detto che per creare una *String* non devi usare la parola chiave *new*. In realtà puoi creare una stringa con il *new*, se davvero ci tieni:

```
String s2 = new String(“abc”);
```

Questa stringa ha lo stesso valore della stringa s che abbiamo creato (senza usare *new*) nel paragrafo scorso. La classe *String* ha infatti un costruttore che prende come parametro un’altra *String*, e assegna alla nuova *String* il valore di quella che le passi – in questo caso la costante *“abc”*. In realtà il fatto di poter creare una *String* senza il *new* è semplicemente una finezza “cosmetica”. Se usi la sintassi senza il *new*, il compilatore aggiunge silenziosamente il *new* che tu hai trascurato. Si tratta semplicemente di una sintassi più breve, che i creatori di Java hanno aggiunto per nostra comodità.
La soluzione dell’*Esercizio 4* è sul CD. Ma tu non la guarderai prima di averci provato da solo, vero? Ci si legge il mese venturo!

Paolo Perrotta

ESERCIZIO 4

Scrivi una classe *Coppia* che contiene due interi. I due interi devono essere passati a ciascun oggetto come parametri del costruttore, e conservati in due campi privati. La classe deve anche avere un metodo *somma()*, che restituisce la somma dei due numeri, e un metodo prodotto(), che ne restituisce il prodotto. Scrivi un semplice programma che testa la classe costruendo un oggetto e stampando sullo schermo il risultato dei vari metodi.

NOTA

OGGETTI E VARIABILI PRIMITIVE

Quando dichiari una variabile in Java, spesso questa variabile è un oggetto.
Ad esempio:

```
Contatore c = new Contatore();
```

Ma come sai ci sono anche variabili che non sono oggetti e quindi non richiedono la parola chiave new.
Ad esempio:

```
int i = 0;
```

Tutte le variabili che non sono oggetti si chiamano variabili primitive del linguaggio.

Le differenze tra oggetti e variabili primitive sono parecchie, e le conoscerai andando avanti con questo corso. Una differenza che probabilmente conosci già sta nella sintassi che usi per manipolare gli oggetti e le variabili primitive.
Per usare gli oggetti devi “mandare loro dei messaggi” attraverso i metodi:

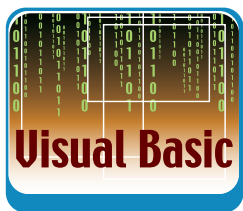
```
c.incrementa();
```

Per usare le variabili primitive, invece, devi usare gli operatori:

```
i = i + 1;
```

Impariamo a costruire Web Services

Un Web Service con MS SOAP toolkit 3.0

In questo appuntamento implementiamo un esempio di web service che utilizza un database Access; parallelamente descriveremo altre caratteristiche del SOAP toolkit.

Nel precedente appuntamento abbiamo presentato il SOAP (Simple Object Access Protocol) toolkit della Microsoft, lo strumento che aggiunge le funzionalità SOAP a Visual Basic 6. Il toolkit è compatibile con le raccomandazioni SOAP 1.1 e le specifiche WSDL 1.1 rilasciate dal W3C. In sintesi abbiamo visto i seguenti argomenti:

1. **WSDL** *(Web Services Description Language)*: un linguaggio XML-based che serve per presentare il web service ed i servizi offerti;
2. **WSML** *(Web Services Meta Language):* un linguaggio che serve per associare alle operazioni descritte nel file *WSDL* i metodi di un componente COM (il file *WSML* è specifico del *Microsoft SOAP toolkit*);
3. alcuni elementi delle API di alto livello del toolkit, in particolare abbiamo utilizzato l'oggetto *SoapClient30* che permette di richiamare i metodi del web service individuati attraverso il file *WSDL;*
4. due client per web service (uno HTML e uno SOAP).

TOOLKIT

Attualmente le specifiche SOAP, oltre che dal SOAP Toolkit della Microsoft, sono implementati da vari tool tra i quali citiamo: *Apache SOAP 2.2* che supporta le specifica SOAP 1.1 ed è basato su Java; *Apache Axis* che è l'evoluzione di Apache SOAP 2.2 e supporta parzialmente le specifiche SOAP 1.2 ed è compatibile con il Toolkit della Microsoft.

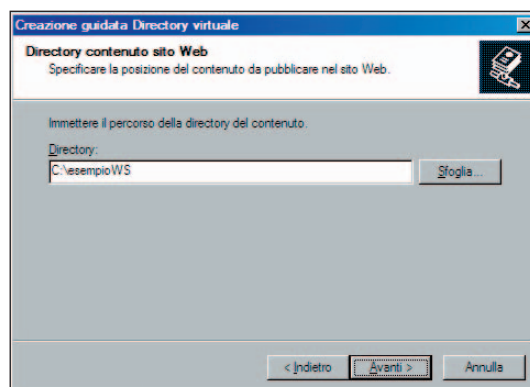



Fig. 1: *Una maschera del Wizard Creazione guidata Directory virtuale.*

In questo appuntamento, invece, implementeremo un web service che fornisce informazioni sugli appartamenti che si affittano in determinate città e vedremo come si costruiscono i messaggi SOAP con le low level API. Inoltre descriveremo come configurare le directory virtuali attraverso lo script *"soapvdir .cmd"*, il generatore di file *WSDL/WSML* e la *Trace Utility.*

IIS, DIRECTORY VIRTUALE E LISTENER ISAPI

Gli esempi che presenteremo sono sviluppati con la piattaforma Windows XP Professional - IIS 5.1. Ricordiamo che in Windows XP Professional, IIS può essere avviato dal pannello di controllo selezionando *"strumenti di amministrazione/Internet Information Services"*. Gli esempi devono essere salvati nella directory *C:/esempioWS/server,* questa bisogna renderla virtuale, attraverso il Wizard *"creazione guidata directory virtuale"* di IIS, con l'alias *"esempioWS"*. Inoltre la directory virtuale bisogna configurarla per l'uso del listener ISAPI *(Internet Server API)*. Per configurare la directory virtuale, nel toolkit, è previsto lo script *soapvdir.cmd.* Per avviare lo script si deve utilizzare il prompt dei comandi impostato sulla directory *C:\programmi\MSSOAP\Binaries* (*MSSOAP* è la directory in cui è installato il toolkit) ed eseguire il seguente comando:

```
soapvdir.cmd UPDATE esempioWS
```

Lo script permette di creare *(Create)* o aggior-

nare *(UpDate)* una directory virtuale; nel nostro caso, dato che la directory viene creata manualmente abbiamo usato l'*UpDate*. Facciamo notare che lo script è necessario anche per configurare le directory delle applicazioni sviluppate con la versione 2.0 del Toolkit altrimenti, installando la versione 3.0, non funzioneranno correttamente. Ora introduciamo il progetto Visual Basic e il database di supporto.

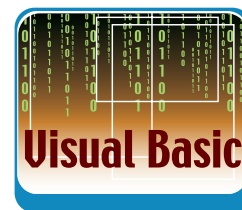
IL PROGETTO VISUAL BASIC

Dato che dobbiamo implementare un componente COM, conviene creare un gruppo di progetti con un progetto EXE e uno DLL Activex. Quest'ultimo lo utilizzeremo per implementare il componente COM mentre il progetto EXE servirà nella fase di progettazione, per testare le funzionalità della DLL e in un secondo momento come client SOAP. Il progetto DLL dovete nominarlo *"esempioWS"* e salvarlo nella sotto directory server. Inoltre, nella scheda generale della finestra delle proprietà del progetto, dovete selezionare le opzioni: *"mantenuto in memoria"* (Retained in Memory) ed *"esecuzione invisibile all'utente"* (Unattanded Execution). Queste impostazioni sono necessarie per assicurare il corretto funzionamento del componente e del tool che genera i file *WSDL/WSML*. Il codice che implementa i servizi del web service va inserito nella classe associata al progetto DLL che dovete nominare *"appartamenti"*. In particolare, nella classe, dovete prevedere due servizi, cioè due metodi, con diverso grado di difficoltà. Il metodo *EchoString* che mostra il contenuto del suo argomento ed il metodo *trovaapp* che fornisce i dati sugli appartamenti in affitto. Questo metodo può essere personalizzato cambiando gli argomenti ed i dati restituiti. In particolare *trovaapp*, riceve una query SQL e restituisce i dati trovati, cioè una stringa in formato XML con gli appartamenti che si affittano (la stringa bisogna "filtrarla" opportunamente, come abbiamo visto nel precedente appuntamento). Di seguito presentiamo le parti principali del codice da inserire nelle due routine.

```
Public Function EchoString(ByVal testString As String) _
    As String
    EchoString = testString
End Function

Public Function trovaapp(ByVal Query As String) As String
   Const adPersistADTG = 0
   Const adPersistXML = 1
   Dim connessione As ADODB.Connection
   Dim rec As ADODB.Recordset
   Dim comXML As MSXML2.DOMDocument
   Set connessione = New ADODB.Connection
   Set rec = New ADODB.Recordset
   Set comXML = New MSXML2.DOMDocument
   connessione.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
   & "Data Source=
         C:\esempioWS\server\dbwebservice.mdb", "", ""
   rec.Open Query, connessione
   rec.Save comXML, adPersistXML
   trovaapp = comXML.xml
   rec.Close
   Set rec = Nothing
   Set comXML = Nothing
   connessione.Close
   Set connessione = Nothing
End Function
```

Il database *dbwebservice.mdb* contiene la tabella *appartamenti* con i seguenti campi: *Id* (di tipo numerico), *citta* (testo), *via* (testo), *prezzo* (testo), *vani* (numerico). Nel progetto EXE dovete referenziare le seguenti librerie: MS XML v4.0; MS Soap Type Library v3.0. Nel progetto DLL, invece, referenziate: MS XML, v4.0 e MS Activex Data Objects 2.7 Library. Naturalmente, quando con il progetto EXE, si provano le funzionalità della DLL bisogna referenziare la *esempioWS.dll*.

WSDL GENERATOR

Tra gli strumenti forniti dal SOAP toolkit c'è il generatore dei file che consentono di identificare i servizi del web service (file *WSDL*) e di associarli ai metodi del componente COM (file *WSML*). Descriviamo come generare i file per il nostro esempio. Dopo aver creato la DLL avviate il tool *WSDL Generator* (da *Start/MS toolkit…/WSDL generator*), questo avvia un wizard che vi guida nelle varie fasi. Le prime due finestre del wizard potete saltarle premendo *next*. Nella terza finestra dovete specificare la DLL e il nome dei file *WSDL* e *WSML*, allora scrivete *"esempioWS"* e scegliete la DLL *esempioWS.dll*, nella finestra successiva (che mostra una struttura *treelike*) selezionate i metodi che volete inserire nel web server, cioè *EchoString* e *trovaapp*. Dopo aver selezionato *next* compare la finestra che permette di selezionare il tipo di *listener* e la sua posizione, il tipo di *listener* è *ISAPI* e la posizione è la seguente *http://nomeserver/esempioWS/server/* dove *esempioWS* è il nome della directory virtuale e *nomeserver* è il nome del vostro server IIS.

W3C

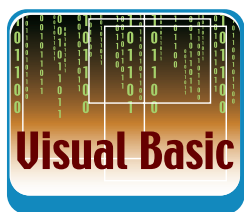
Attualmente il W3C si sta occupando della creazione dello standard SOAP 1.2, le sue specifiche verranno divise in due parti: SOAP messaging http://www.w3.org/TR/soap12-part1/ **e SOAP-RPC-http** http://www.w3.org/TR/soap12-part2/

Nella maschera successiva invece si possono inserire gli URI che saranno usati nel file *WSDL*, lasciate i valori di default e selezionate *next*. Nella maschera successiva si può scegliere il tipo di caratteri *Unicode*, tra *UTF-8* (default) e *UTF-16* e dove salvare i file che verranno creati, anche in questo caso possiamo lasciare i valori di default, selezionate *next* e poi *finish* per avviare la generazione.

Fig. 2: Il WSDL generator.

Facciamo notare che nella directory prescelta sono generati quattro file. Inoltre notate che nel file *esempioWS.WSDL* la scelta del tipo di *listener* definisce il valore dell'elemento *location* (che si trova nel tag service), in particolare nel caso di *listener ISAPI* abbiamo:

```
location='http://nomeserver/esempioWS/server/
                                    esempioWS.WSDL.
```

Mentre nel caso di listener ASP avremo

```
location='http://nomeserver/esempioWS/server/
                                    esempioWS.ASP
```

Nel caso di ISAPI il listener è il file *soapisap.dll* mentre nel caso ASP è il file *esempioWS.ASP* che sicuramente può essere facilmente personalizzato. Le informazioni sulla location le utilizzeremo quando descriveremo la *Trace Utility*. Ora vediamo il codice da inserire nel form del progetto *client*.

IL CLIENT SOAP

Nel precedente articolo abbiamo presentato un client SOAP che con l'oggetto *SoapClient30* (dell'API di alto livello) si collega a dei web service pubblicati su internet. Ora realizzeremo un client che si collega al web service *"esempioSW"* attraverso gli elementi dell'API di basso ed alto livello. Iniziamo utilizzando le API di alto livello, in particolare in un pulsante inseriamo il codice che permette di inviare una query SQL al servizio *"trovaapp"*, la query da inviare è la seguente:

```
Select * from appartamenti where citta="milano" and
                                        prezzo="500"
```

Cioè seleziona tutti gli appartamenti che si affittano nella città di Milano con prezzo dell'affitto uguale a 500 euro.

```
Private Sub esempioalto_Click()
 Dim soapClient3 As MSSOAPLib30.SoapClient30
 Set soapClient3 = New MSSOAPLib30.SoapClient30
 Call soapClient3.MSSoapInit _
 ("http://localhost/esempioWS/ server/esempioWS.WSDL")
 If Err <> 0 Then
  MsgBox "errore di inizializzazione" + Err.Description
 End If
 RichTextBox1 = soapClient3.trovaapp _
 ("select * from appartamenti where citta=""milano""
                                and prezzo = ""500""")
 If Err <> 0 Then
  MsgBox Err.Description
 End If
 Set soapClient3 = Nothing
End Sub
```

Sul form dovete prevedere il *RichTextBox1* per mostrare la stringa restituita dal metodo *trovaapp*.

SPAZIO NOMI

Lo spazio dei nomi è un "identificatore" univoco per gli elementi presenti in un documento XML e serve per evitare conflitti di nomi. Lo spazio nei nomi tempuri.org, che ritroviamo nei documenti SOAP, è temporaneo e dovrebbe essere utilizzato soltanto in fase di sviluppo e personalizzarlo (magari con il nome dell'azienda che fornisce il servizio) prima che il web service sia reso pubblico.

LE LOW-LEVEL API

Il SOAP toolkit per elaborare e modificare i messaggi SOAP fornisce diversi elementi di basso livello. In particolare, sul lato client, possiamo usare l'oggetto *ISoapConnector* (implementato da *Httpconnector30*) per impostare una connessione e per avviare le azioni legate ai messaggi SOAP, l'oggetto *SoapSerializer30* per costruire i messaggi e l'oggetto *SoapReader30* per leggere le risposte inviate dal server. Ora, costruiamo un semplice esempio utilizzando questi oggetti, in particolare invochiamo il servizio *EchoString* con il codice seguente.

```
Private Sub conserializzazione_Click()
 Const Metodo = "EchoString"
 'il metodo invocato
```

```
Const arg = "Hello world"
'il valore dell'argomento TestString
Const SoapAction = _
"http://tempuri.org/esempioWS/action/appartamenti."
Const END_POINT_URL = _
"http://localhost/esempioWS/server/esempioWS.WSDL"
Const MESS = "http://tempuri.org/esempioWS/message/"
Dim Serializer As SoapSerializer30
Dim Reader As SoapReader30
Dim Connector As SoapConnector30
Set Connector = New HttpConnector30
Connector.Property("EndPointURL") = END_POINT_URL
Connector.Connect
Connector.Property("SoapAction") = SoapAction
                                            & Metodo
Connector.BeginMessage
 Set Serializer = New SoapSerializer30
 Serializer.Init Connector.InputStream
  Serializer.StartEnvelope
   Serializer.StartBody
    Serializer.startElement Metodo, MESS
     Serializer.startElement "TestString"
      Serializer.WriteString arg
     Serializer.endElement
    Serializer.endElement
   Serializer.EndBody
  Serializer.EndEnvelope
Connector.EndMessage
Set Reader = New SoapReader30
Reader.Load Connector.OutputStream
If Not Reader.Fault Is Nothing Then
 MsgBox Reader.FaultString.Text, vbExclamation
Else
 Me.RichTextBox1 = Reader.RpcResult.Text
End If

End Sub
```

La procedura precedente: stabilisce una connessione con il server; costruisce un messaggio SOAP che incapsula l'invocazione del metodo *EchoString* ed elabora il messaggio restituito dal server. La connessione verso l'URL, specificato in *End_Point_URL*, è creata con l'oggetto *Httpconnector30*, con i suoi metodi si impostano, anche, lo scopo del messaggio *(SOAPAction)*. Facciamo notare che i valori di *End_ Point_URL*, *SOAP Action* e *MESS* sono contenuti nel file *WSDL*. Inoltre, attraverso le proprietà del *Httpconnector30* si definisce anche lo *Stream* dove verranno spediti i messaggi.
Dopo questa fase d'impostazione vengono create le parti del messaggio SOAP cioè *Envelope* e *Body* (che abbiamo descritto nel precedente appuntamento) e attraverso i metodi *StartElement*, *EndElement* e *WriteString*, vengono inserite le informazioni sul servizio invocato e sui suoi parametri.

TRACE UTILITY

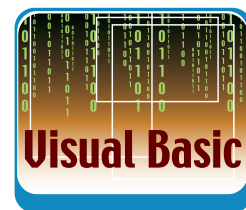
La *Trace Utility*, o meglio *TCP/IP trace utility*, serve per vedere i messaggi SOAP che transitano su http, durante il "dialogo" tra client e server. Per utilizzarla, bisogna modificare il codice precedente, impostando la porta 8080 nell'URI che individua il file *esempioWS.WSDL* cioè *http://nomeserver:8080/esempioWS/server/esempioWS. WSDL*. Nella *Trace Utility*, invece, bisogna selezionare la voce di menu *"file/New/formatted trace"*. Nella maschera che compare, specificare il nome del server IIS (nel *textbox Destination host*) e premere *ok*. Le azioni precedenti, sul *Trace Utility*, fanno comparire una finestra divisa in tre parti dove, quando si utilizza il web server *esempioWS*, verranno mostrati i messaggi SOAP. In particolare in Fig. 3 mostriamo la *Trace Utility* con due messaggi SOAP: la richiesta del client e la risposta del server.

```
MSSoapT - [Listening for    na at port 8080, destination port 80]
File  View  Window  Help
169.254.1.8
169.254.1.8
  Message # 1
  Message # 2

<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body>
    - <SOAPSDK4:EchoString
        xmlns:SOAPSDK4="http://tempuri.org/esempioWS/message/">
        <testString>Hello World</testString>
      </SOAPSDK4:EchoString>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>

- <SOAP-ENV:Envelope
    xmlns:SOAPSDK1="http://www.w3.org/2001/XMLSchema"
    xmlns:SOAPSDK2="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:SOAPSDK3="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  - <SOAP-ENV:Body SOAP-
      ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    - <SOAPSDK4:EchoStringResponse
        xmlns:SOAPSDK4="http://tempuri.org/esempioWS/message/">
        <Result>Hello World</Result>
      </SOAPSDK4:EchoStringResponse>
    </SOAP-ENV:Body>
  </SOAP-ENV:Envelope>
```

Fig. 3: *Il Trace Utility che mostra due messaggi SOAP.*

CONCLUSIONI

Nei due appuntamenti dedicati al SOAP, data la vastità dell'argomento, non abbiamo potuto descrivere le caratteristiche avanzate degli strumento utilizzati, per questo vi consigliamo di approfondire lo studio del protocollo SOAP e del toolkit, magari sviluppando altri esempi inerenti la gestione dei tipi complessi e degli attachment.
Buon lavoro!

Massimo Autiero

Un esempio applicativo di Struts

Struts: un portale in Java

Questo è il primo di una serie di articoli che ci porterà a conoscere le potenzialità della programmazione tramite framework. Realizzeremo un portale utilizzando il framework open source Struts.

La realizzazione di qualsiasi applicazione Web professionale, dalle più semplici fino ai portali verticali più complessi, deve obbligatoriamente seguire le regole dello sviluppo "di qualità", sia dal punto di vista della progettazione sia per quanto concerne la pulizia e la comprensibilità del codice che scriviamo. A partire da questo articolo, il primo di una serie, realizzeremo un portale verticale con molte delle funzionalità che ci si aspetta di avere in realizzazioni professionali, progettazione UML prima di iniziare la realizzazione e cercando di mantenere il nostro codice ad un livello qualitativo che si avvicini a quello delle *Code Conventions for the Java™ Programming Language*, le linee guida stabilite da Sun Microsystem per la realizzazione di codice Java di buona qualità. Il nostro obiettivo di realizzazione è un portale verticale, dotato quindi di molti servizi tipicamente forniti dai portali, sviluppato utilizzando la tecnologia Java 2 Enterprise Edition.

***Fig. 1:* La Home Page del Portale che svilupperemo.**

ARCHITETTURA FUNZIONALE

In Fig. 1 potete vedere in anteprima la Home Page

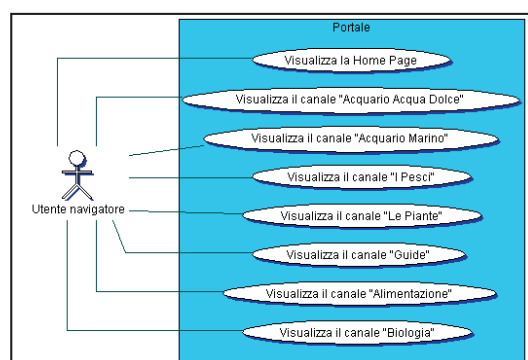



***Fig. 2:* Use Case Diagram dell'utente navigatore.**

del portale che andremo a realizzare. Siamo in questo caso in una situazione privilegiata in quanto è già disponibile, prima ancora di iniziare a studiare l'architettura del portale, una bozza di *look&feel*, cioè della componente grafica che dovremo erogare con la nostra applicazione. Dal punto di vista puramente applicativo, la componente grafica potrebbe essere, almeno all'inizio. Un fatto trascurabile, ma il poter sapere in anticipo da quali sezioni è composta la pagina, la loro forma ed alcuni loro comportamenti ci può dare immediatamente qualche indicazione, ad esempio, su quali tecnologie utilizzare per la composizione della pagina e su quale livello di template usare. Dopo aver intervistato il cliente ci sono chiare le funzionalità che il nostro portale deve offrire all'utente navigatore tradizionale, in particolare questa tipologia di utenza può navigare sul portale e visualizzare la home page e tutte le pagine secondarie, cioè quelle che chiamiamo "pagine di canale". Questo caso d'uso è descritto con precisione in Fig. 2, dove viene proposto il primo *Use Case Diagram* che abbiamo ricavato durante la nostra raccolta dei requisiti. Un discorso a parte merita la profilatura degli utenti che, come si vede nello *Use Case Diagram* dedicato visibile in Fig. 3, consiste

Fig. 3: ***Use Case Diagram dei profili di utenza.***

fondamentalmente nell'avere due classi di utenti: i navigatori e gli utenti registrati. L'utente registrato è una specializzazione del generico utente navigatore e mentre quest'ultimo ha solo la facoltà di effettuare la registrazione mediante l'apposita funzionalità presente nel box di login, l'utente registrato può accedere al suo profilo attraverso la procedura di login, richiedere una nuova password se l'ha dimenticata ed effettuare il logout. Per definire meglio quali possano essere le funzionalità riservate agli utenti registrati è necessario scrivere un nuovo *Use Case Diagram*. In particolare sappiamo che l'utente registrato può:

- modificare il proprio profilo,
- ricevere la newsletter,
- visualizzare eventuali aree registrate.

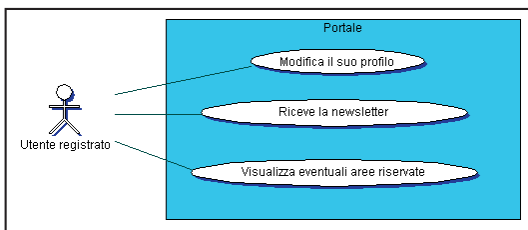



Fig. 4: ***Use Case Diagram dell'utente registrato.***

Questo comportamento funzionale è descritto nel diagramma di Fig. 4. Adesso che abbiamo messo un po' di carne al fuoco è bene iniziare a pensare ad un'architettura tecnica che possa essere compatibile con i requisiti funzionali che abbiamo raccolto intervistando il nostro ipotetico cliente virtuale.

UTILIZZO DI UN FRAMEWORK

Nello sviluppo di applicazioni Web Based, in generale, è molto raro che si parta dal foglio bianco reinventando ogni volta tutte le componenti applicative di base. Più spesso si parte da framework o da semilavorati che consentono di avere già implementati tutta una serie di servizi comuni che siano adeguatamente collaudati e supportati e che mantengano nel tempo un'interfaccia standard che ci consenta un aggiornamento delle funzionalità senza dover stravolgere il codice della nostra applicazione.
In particolare un framework deve fornire un modello di sviluppo del software standardizzato e che nel tempo sia consistente. Il punto di forza che consente ad un framework di essere realmente conveniente è la gestione autonoma di un insieme di servizi generici presenti in qualsiasi tipologia di applicazione Web, come ad esempio la gestione dell'interfaccia utente, l'astrazione del livello di memorizzazione dei dati all'interno di un database, la gestione della configurazione in appositi file separati dall'applicazione. Un framework inoltre contiene (e di conseguenza fornisce nativamente al progettista ed allo sviluppatore) alcuni strumenti, metodi e best practices per accelerare lo sviluppo mantenendo un elevato standard qualitativo della realizzazione. Ecco che quindi iniziano a delinearsi i vantaggi che si possono avere nell'utilizzare un framework consolidato piuttosto che doversi scrivere da zero tutte le componenti, anche quelle più ripetitive. Ai framework possiamo delegare tutti i compiti ripetitivi o consolidati che sono comuni a qualsiasi applicazione Web e noi possiamo concentrarci quasi esclusivamente sulle problematiche di business che la nostra applicazione deve gestire.

STRUTS

Visti gli innumerevoli vantaggi derivanti dall'utilizzo di un Application Framework nella realizzazione di applicazioni Web Based, il panorama di questi oggetti software fornisce un numero molto elevato di possibilità, da quelle open source a quelle commerciali. Per la realizzazione del nostro portale sull'acquariofilia utilizzeremo *Struts* (parte del progetto Jakarta), un framework open source scritto utilizzando Servlet e JSP e quindi basato interamente sulla tecnologia Java 2 Enterprise Edition. Si tratta di un framework semplice e snello ma dalle caratteristiche molto interessanti. La prima tra tutte è la sua architettura interna che è completamente basata sul pattern *Model View Controller* che, come sappiamo, consente di separare tra loro le componenti applicative: il *Model* che implementa le funzionalità di business, la componente di *View* che implementa la logica di presentazione ed il *Controller* che implementa la logica di controllo. Per utilizzare *Struts*, pertanto, dovremo solo decidere a priori la navigazione all'interno del portale descrivendola sotto forma di *"actions"*, azioni che vengono eseguite sull'applicazione, e rimapparla all'interno di un file di configurazione che assocerà tutte queste azioni a componenti software dedicate e ne descriverà il percorso di navigazione. Come si vedrà in seguito tutto questo è più facile da fare che da descrivere.

I DIAGRAMMI UML

UML si compone di una serie di diagrammi che permettono di definire e progettare un sistema nella sua interezza. I diagrammi a disposizione sono:

Use Case Diagram
Class Diagram
Component Diagram
Deployment Diagram
State Chart Diagram
Activity Diagram
Sequence Diagram
Collaboration Diagram

Esistono poi diagrammi fuori dallo standard UML che consentono di descrivere altri aspetti del sistema, come ad esempio:

Entità Relationship Diagram
per definire le entità del sistema e le relazioni tra esse

Web Application Diagram
per definire le architetture delle applicazioni web.

PROGETTARE LA NAVIGAZIONE

Per descrivere la navigazione che intendiamo implementare utilizziamo un *Activity Diagram UML* (visibile in Fig. 5), cioè uno strumento che ci permette di definire una serie di stati, le azioni che portano da uno stato all'altro e delle condizioni che consentono di modificare il comportamento di un'azione in funzione di condizioni al contorno. Nel nostro caso abbiamo questa correlazione di oggetti:

- **stati**: sono le pagine web erogate alle diverse tipologie di utenza;
- **azioni**: sono le classi Java che implementano le logiche di business associate a ciascuna visualizzazione;
- **condizioni**: sono le discriminanti che, all'interno di una singola azione, ci consentono di decidere quale stato raggiungere, cioè quale pagina JSP visualizzare.

Nella nostra implementazione, che utilizza il motore del *framework Struts*, possiamo essere ancora più precisi ed identificare ciascuna componente con un elemento del framework:

- **stati:** sono le pagine JSP che implementano le viste che vogliamo erogare alle diverse tipologie di utenza;
- **azioni:** sono le classi Java, derivate dalla classe più generale *Action*, che implementano il passaggio dallo stato precedente alla pagina JSP di destinazione associata alla visualizzazione;
- **condizioni:** sono gli eventi che si verificano all'interno di una particolare azione, in particolare in caso di errore il controllo passerà ad una pagina di gestione dell'errore stesso, in caso contrario si potrà passare alla visualizzazione della pagina JSP corretta.

IL PATTERN MVC

In genere, e bene che le componenti di un'applicazione siano sufficientemente separate tra loro e che possano essere sostituite con altre implementate diversamente a condizione che ne rispettino l'interfaccia. Il pattern MVC consente di separare tra loro le componenti applicative: il Model che implementa le funzionalità di business, la componente di View che implementa la logica di presentazione ed il Controller che implementa la logica di controllo.

Fig. 5: L'Activity Diagram della navigazione nel Portale.

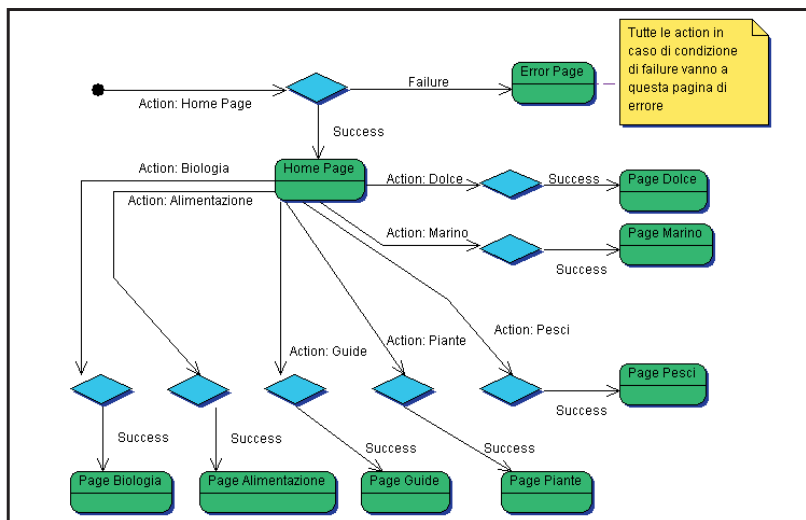
Il diagramma delle attività di Fig. 5, pertanto, descrive l'accesso alla Home Page attraverso la chiamata ad una particolare *Action* che in caso di "success" mostra effettivamente la Home Page, in caso di "failure", (per esempio perché non è possibile contattare il database che contiene i dati da pubblicare nella pagina), passa il controllo ad una pagina di errore. Diagramma indica anche che questa gestione delle condizioni di errore è comune a tutte le action e che non viene replicata esplicitamente per non appesantire inutilmente il diagramma stesso. In casi più complessi, o dove non esista una regola precisa da poter descrive, sarà necessario indicare sul diagramma delle attività tutti i possibili percorsi di navigazione.

DESCRIVERE LA NAVIGAZIONE

All'interno di ciascuna applicazione basata su *Struts*, nella directory WEB-INF è presente il file *struts-config.xml* che contiene la configurazione di tutta l'applicazione dal punto di vista architetturale. La navigazione è una delle componenti fondamentali di questo tipo di configurazione, pertanto all'interno di questo file dovremo andare a descriverla con precisione. Nel file di configurazione troveremo la sezione *<action-mappings />* deputata proprio a contenere il mapping della azioni, cioè il comportamento che l'applicazione deve avere in funzione delle richieste dell'utente. Inizieremo quindi a descrive la prima di queste azioni, quella che riguarda la Home Page, in questo modo:

```
<action path="/home"
    scope="request"
    type="com.fish.actions.HomeAction"
    validate="false">
    <forward name="success" path="/pages/home.jsp"/>
</action>
```

Stiamo quindi dicendo al framework che quando l'utente seleziona l'url: *http://nomeserver/applicazione/home.do* il controllo applicativo deve essere passato alla classe java *com.fish.actions.HomeAction* e se e solo se questa classe "restituisce il valore *success*" allora il controllo deve essere passato alla pagina */pages/home.jsp/*. A questo punto dovrebbe essere abbastanza chiaro come sia possibile descrivere tutta la navigazione della nostra applicazione attraverso frammenti di codice XML di questo tipo:

```
<action
    path="/actionname"
    scope="request"
    type="com.fish.actions.ActionClassName"
```

```
    validate="false">
    <forward name="success" path=
"/pages/pagetogo.jsp"/>
</action>
```

C'è da notare come, all'interno di ogni singola *action,* sia stato descritto il percorso da seguire nel caso tutto vada bene, ma non nel caso ci sia un errore di qualche tipo. Questo dipende dal fatto che la gestione degli errori è centralizzata ed abbiamo stabilito che in caso di "failure" di qualsiasi *action* il controllo debba passare ad una pagina di errore. Questo si realizza inserendo nel file *struts-config.xml* la seguente sezione

```
<global-forwards>
    <forward name="failure" path=
                    "/pages/errorpage.jsp"/>
</global-forwards>
```

Si tratta di una sezione che definisce i percorsi a livello più generale. Se una classe restituisce "failure" ma non descrive il comportamento per questo evento il forward viene cercato nella sezione di più alto livello *global-forwards* e questo consente, per esempio, di evitare di scrivere la stessa gestione degli errori nella definizione di tutte le *action*.
A questo punto però è necessario fare un po' di chiarezza su alcuni aspetti. Innanzi tutto si è parlato di un url nel formato *http://nomeserver/applicazione/home.do,* ma qual è il significato del suffisso *".do"*?
È presto detto: all'interno del file *web.xml* dell'applicazione, anch'esso contenuto nella directory *WEB-INF,* è presente la seguente sezione:

```
<servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Si tratta dell'assegnazione di un url-pattern alla servlet identificata con il nome mnemonico action.
La servlet in questione è naturalmente il controller di *Struts,* si tratta infatti della classe *org.apache.struts.action.ActionServlet,* ed il mapping prevede che per referenziare il parametro principale della servlet si possa utilizzare, appunto, il pattern *.do.
Questo significa che invece di scrivere un url di questo genere: *http://nomeserver/applicazione/action?action=home* siamo nelle condizioni di poter scrivere: *http://nomeserver/applicazione/home.do.*
Altro punto su cui è necessario fare chiarezza è la restituzione di un valore da parte di una classe.
In realtà, non si tratta, di una vera e propria restituzione di valore come la si potrebbe intendere dal punto di vista funzionale, quanto piuttosto di un messaggio che la classe Java deve inviare ad un altro oggetto del sistema prima di terminare l'esecuzione. In particolare, una classe che implementa una singola *action* deve soddisfare alcuni requisiti:

- deve estendere la classe *org.apache.struts.action.Action* o una sua sottoclasse;
- deve implementare il metodo *execute();*
- deve restituire un oggetto istanza della classe *ActionForward*;
- normalmente lo restituisce attraverso il metodo *mapping.findForward(messaggio)*; dove il messaggio può essere success, failure oppure un messaggio personalizzato.

Ecco quindi spiegato il significato di far ritornare un valore alla classe, si tratta in realtà di utilizzare il metodo *execute()* e di fargli ritornare il messaggio che preferiamo, sia esso success, failure oppure un messaggio che descrive meglio l'evento. Naturalmente questo messaggio dovrà essere mappato correttamente all'interno del file di configurazione di *Struts* per descrivere quale pagina JSP dovrà essere utilizzata per il rendering della risposta da inviare al browser web.

LE CLASSI PER LA HOME PAGE

Come tutte le architetture mediamente complesse, anche li nostro portale ha bisogno che molte componenti vengano messe a punto prima di poter ottenere anche solo la Home Page. Ci accorgeremo però successivamente che, una volta creata l'architettura di base, sarà molto semplice e ci costerà molto poco aggiungere canali e servizi. Quello che ci manca adesso, dopo aver configurato correttamente il framework, è la classe che implementa la Home Page. Tutte le classi di business della nostra applicazione sono contenute nell'alberatura:

```
.../WEB-INF/src/com/fish/actions
```

Quelle che ci servono per iniziare sono le seguenti:

- **InitAction:** viene invocata al primo accesso all'applicazione e consente di inizializzare alcuni oggetti applicativi importanti come una sessione HTTP o un Javabean da mettere in sessione che ci consentirà di scambiare informazioni tra le varie componenti dell'applicazione
- **BaseAction:** è una classe che viene definita come sottoclasse della *Action* tradizionale di *Struts,* vale a dire la *org.apache.struts.action. Action.* Noi la useremo come classe di base da cui tutte le altre action erediteranno, in modo da centralizzare eventuali comportamenti comuni a tutte le action.
- **HomeAction:** è la classe che implementa real-

IL FRAMEWORK STRUTS

Struts è un applciation framework open source scritto utilizzando Servlet e JSP e quindi basato interamente sulla tecnologia Java 2 Enterprise Edition. Si tratta di un framework semplice e snello ma dalle caratteristiche molto interessanti, la sua architettura interna è completamente basata sul pattern Model View Controller che consente di separare tra loro le componenti applicative. E' interamente configurabile dall'esterno attraverso l'utilizzo di file di configurazione XML che consentono di descrivere la struttura di navigazione del portale, le classi che devono essere eseguite in funzione delle scelte di navigazione dell'utente e le componenti view che si devono occupare del rendering dell'informazione. Struts è parte del progetto Jakarta ed è scaricabile dall'url: http://jakarta.apache.org/struts/

CODE CONVENTIONS FOR THE JAVA‰ PROGRAMMING LANGUAGE

Si tratta di linee guida sintattiche promosse direttamente da Sun Microsystem e che hanno l'obiettivo di indicare come deve essere scritto il codice Java per poter essere immediatamente comprensibile anche da chi non lo ha direttamente scritto. L'utilizzo di convenzioni, in generale, è una buona abitudine che permette a chi lavora in gruppo di abbassare i problemi di comprensione del codice scritto da altri. Altro vantaggio di un approccio rigido alle convenzioni è la possibilità di manutenere con più facilità il codice scritto anche a distanza di molto tempo. Tutto questo si traduce in una implicita qualità sintattica del codice prodotto.

mente la logica di business necessaria a produrre la Home Page, si occupa di reperire le informazioni da pubblicare, presumibilmente su un database, e di inserirle in un bean contenitore in modo che queste siano lette dalla pagina JSP di destinazione.

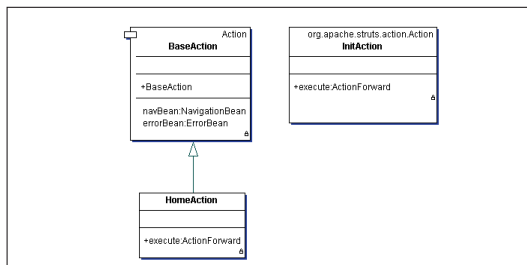



Fig. 6: ***Class Diagram del package actions.***

In Fig. 6 è visibile il *Class Diagram* del package actions che descrive le tre classi necessarie alla produzione della Home Page del portale.
Ed ecco il codice della classe *HomeAction* che implementa la logica di business per la produzione della Home Page:

```
package com.fish.actions;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;
/**
 * @author Masssimo Canducci<br><br>
 *
 * Questa classe implementa la logica di business per
                                        la action /home<br>
 * che consente di mostrare all'utente la Home Page
                                        del Portale<br><br>
 */
public class HomeAction extends
                              com.fish.actions.BaseAction {
   /**
    * @author Masssimo Canducci<br><br>
    * @return success / failure<br><br>
    *
    */
   public ActionForward execute(ActionMapping mapping,
         ActionForm form, HttpServletRequest request,
                        HttpServletResponse response)
      throws Exception {
      try {
         // reperimento delle informazioni ed
                                    inserimento in un
         // javabean per il trasporto alla JSP di
                                         destinazione
         return mapping.findForward("success"); }
      catch (RuntimeException e) {
         System.out.println(e.toString());
            return mapping.findForward("failure"); } }
}
```

Quello che fa la nostra action è semplicemente implementare il metodo *execute()* che si deve occupare di reperire le informazioni da pubblicare in HomePage e, se tutto è andato bene, di restituire un oggetto *ActionForward* attraverso l'istruzione:

```
return mapping.findForward("success");
```

Nel caso in cui, invece, qualcosa sia andato storto nel reperimento delle informazioni, il controllo dell'esecuzione passerà immediatamente al blocco *catch* che si occuperà di scrivere l'errore nella console e di restituire il messaggio di *failure* attraverso l'istruzione:

```
return mapping.findForward("failure");
```

Queste due condizioni sono state censite nel file di configurazione del framework, pertanto *Struts* sarà in grado, senza doverlo indicare esplicitamente nel codice della classe che implementa la logica di business, di indirizzarci vero la pagina JSP corretta in funzione del messaggio restituito dalla action.

CONCLUSIONI

La realizzazione di un portale non è cosa semplice, quello che abbiamo visto in questo articolo è buona parte dell'architettura necessaria per la messa online di un portale completo. Siamo partiti dall'analisi dei requisiti per scrivere qualche diagramma UML che descrivesse funzionalmente i casi d'uso più macroscopici. Siamo poi passati alla definizione della struttura di navigazione del portale che ci ha consentito di configurare adeguatamente l'application framework che stiamo utilizzando. Infine abbiamo realizzato la classe che implementa la logica di business necessaria a pubblicare la Home Page del portale più qualche classe trasversale come quella di inizializzazione e quella di base. A questo punto abbiamo quasi tutto quello che ci serve per vedere online il nostro portale, ci mancano ancora i bean per trasportare le informazioni e le pagine JSP per presentarle. A proposito di questo, ci troveremo di fronte ad un problema: nella configurazione del framework dobbiamo indicare la pagina di destinazione per ogni messaggio. Come è possibile fare in modo che ciascuna delle pagine di destinazione contenga il menù, la testata e la spalla di destra senza dover replicare queste componenti in ogni singola pagina del portale? Lo faremo nel prossimo articolo arricchendo il framework e realizzando un controller di secondo livello.

Massimo Canducci

Tecniche avanzate di dialogo fra classi

Delegati in C#

Una delle caratteristiche più interessanti introdotte con il framework .NET è l'uso dei delegati e degli eventi: ma i delegati non nascono dal nulla...

I programmatori C e C++ conoscono bene una tecnica utilissima per far sì che sia possibile specificare funzioni da richiamare al verificarsi di un evento. Lo sviluppatore Win32 utilizza spesso le funzioni callback per permettere a Windows di notificare cambiamenti di stato, esito di enumerazioni, avanzamenti di operazioni. I puntatori a funzione C non sono però type-safe: sono infatti semplici puntatori che puntano a locazioni di memoria, non specificano la segnatura della funzione che andranno a chiamare, né sono orientati agli oggetti. I delegati sono, al contempo, fortemente type-safe e orientati agli oggetti.

NOTIFICA SENZA DELEGATI

Ovvero: ci lasci il suo indirizzo: la chiamiamo noi… Vediamo una tecnica che potremo utilizzare per effettuare, in C#, la notifica di eventi. Supponiamo di avere una classe che esegue un calcolo piuttosto lungo, e di voler far sì che questa possa notificare il progresso del calcolo a chi sia interessato, senza utilizzare i delegati. La nostra classe (che simulerà un calcolo particolarmente complesso) sarà la seguente

```
class public class ClasseCalcolo
  { public ClasseCalcolo()
    { }
    public void SimulaUnCalcoloLungo()
    {  int risultato=0;
       for (int a=0;a<100;a=a+20)
       { risultato++;
          System.Threading.Thread.Sleep(1000);  } }
}
```

un primo metodo per far sì che un oggetto istanza di un'altra classe, per esempio *Logger*, possa ricevere notifiche sull'andamento del calcolo potrebbe essere quello di passare tale istanza alla *ClasseCalcolo*, e far chiamare un metodo predefinito nella funzione *SimulaUnCalcoloLungo()*. Per essere sicuri che il metodo sia presente, e per generalizzare l'approccio, potremmo definire una interfaccia cui gli oggetti che vogliano essere notificati debbano aderire:

```
interface INotificatore
{ public bool NotificaAvanzamento(int percentuale); }
public class Logger:INotificatore
{  public bool NotificaAvanzamento(int percentuale)
   { System.Console.WriteLine("Avanzamento:
                                        {0}%\n",percentuale);
      return true; }
}
```

dovremmo poi riscrivere il nostro metodo di *ClasseCalcolo* inserendo la chiamata all'oggetto di notifica:

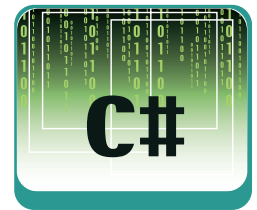
```
public void SimulaUnCalcoloLungo(INotificatore Notif)
      {  int risultato=0;
         for (int a=0;a<100;a=a+10)
         {    risultato++;
              if (Notif!=null) Notif.NotificaAvanzamento(a);
              System.Threading.Thread.Sleep(1000);  } }
```

nel codice che utilizzerà le nostre classi dovremmo scrivere:

```
Logger myLogger= new Logger();
ClasseCalcolo myCalc=new ClasseCalcolo();
myCalc.SimulaUncalcoloLungo(myLogger);
```

Questa tecnica ha però molti svantaggi, per esempio il fatto che se volessimo inserire un altro punto di notifica dovremmo cambiare l'interfaccia cui l'oggetto chiamato deve appartenere. Se poi più oggetti dovessero essere notificati, magari in punti diversi della nostra funzione, le cose si complicherebbero ulteriormente. Inoltre non è possibile utilizzare questo metodo così com'è per chiamare sia metodi statici che metodi d'istanza.

ARRIVANO I DELEGATI!

Ad una prima analisi, un delegato è una rappresentazione, un involucro di un metodo, una sorta di

SCOPO DELL'ARTICOLO

"I delegati rendono possibili scenari che altri linguaggi hanno gestito con i puntatori a funzione. Comunque, a differenza dei puntatori a funzione, i delegati sono orientati agli oggetti e sicuri sul tipo". In questo articolo verranno introdotti i delegati e il loro uso in C#. Ne verranno illustrate le tecniche d'uso sincrone e asincrone. Verranno poi introdotti gli eventi come particolare e utile caso di utilizzo di delegati.

puntatore a funzione che però specifica il tipo (la segnatura) della funzione cui punta. In realtà, come vedremo, è molto di più: è un'istanza di una classe creata su misura! Riprendiamo il nostro esempio della classe *ClasseCalcolo* utilizzando un delegato. I passi per utilizzare un delegato sono quattro:

1) definire il tipo delegato per la segnatura di metodo voluta (in realtà la classe delegato).
2) creare un oggetto di questo tipo.
3) creare uno o più gestori di delegato.
4) assegnare al delegato il gestore o i gestori creati.

la sintassi per dichiarare un delegato è:

```
[attributi] [modificatori] delegate tipo_risultante
                        identificatore ([parametri formali]);
```

dove fra parentesi quadre sono riportati elementi opzionali. Nel nostro esempio potremo definire due classi di delegati: una fornirà notizie sul progresso e un'altra fornirà notizie sulla fine del calcolo e sul risultato.

```
public delegate bool DelegatoNotifica (int percentuale);
public delegate bool DelegatoFineCalcolo (int
                              risultato, DateTime OraFine);
public class ClasseCalcolo
{   public DelegatoNotifica objDelegatoNotifica=null;
    public DelegatoFineCalcolo
                              objDelegatoFineCalcolo=null;
    public ClasseCalcolo()
    { }
    public void SimulaUnCalcoloLungo()
    {  int risultato=0;
       for (int a=0;a<100;a=a+20)
       { if (objDelegatoNotifica!=null)
                                   objDelegatoNotifica(a);
          risultato++;
          System.Threading.Thread.Sleep(1000);  }
       if (objDelegatoFineCalcolo!=null)
        objDelegatoFineCalcolo(risultato, DateTime.Now); }
}
```

Fig. 1: Diagramma di derivazione di un delegato.

È possibile ridefinire i metodi di aggiunta/rimozione degli eventi ridefinendo entrambi gli accessor add e remove. Questa tecnica può essere utile per tener conto, anche attraverso l'uso della Reflection, di quanti e quali gestori sottoscrivono i nostri eventi. Se non si ridefiniscono entrambi gli accessor, si incorre in un errore di compilazione (CS0065). Per notizie su questa tecnica, vedere qui: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/csref/html/vclrfeventpg.asp

per comprendere l'uso degli oggetti delegati che abbiamo fatto nel codice occorre spiegare cosa succede quando viene dichiarato un delegato. Il compilatore infatti trasforma la riga:

```
public delegate bool DelegatoNotifica (int percentuale);
```

in una dichiarazione di classe derivata da *System.MulticastDelegate*, che a sua volta deriva da *System.Delegate*. Questo è quello che si vede utilizzando un disassemblatore come *ILDASM* o *.NET Reflector*:

```
public class sealed DelegatoNotifica
                          :System.MulticastDelegate
{ public DelegatoNotifica(object object, IntPtr method);
  public virtual IAsyncResult BeginInvoke(int
    percentuale, AsyncCallback callback, object object);
  public virtual bool EndInvoke(IAsyncResult result);
  public override bool Invoke(int percentuale); }
```

disassemblando poi il codice prodotto dal compilatore per quanto riguarda la funzione *SimulaUnCalcoloLungo* vediamo che le chiamate ai vari delegati altro non sono che la chiamata al metodo *Invoke* degli oggetti istanze della classe appena vista, ovvero:

```
if (objDelegatoNotifica!=null) objDelegatoNotifica(a);
```

diventa questo: (*num2* è il parametro a)

```
if (this.objDelegatoNotifica != null)
{ this.objDelegatoNotifica.Invoke(num2); }
```

vediamo come utilizzare questi delegati. Creeremo una classe *Logger* che stampa a video la percentuale di progresso:

```
public class Logger
{ string _id;
   public bool NotificaAvanzamento(int a)
   { Console.WriteLine ("Notificatore {0}
                         \tavanzamento {1}%",_id,a);
      return true; }
   public static bool NotificaAvanzamentoStatico(int a)
   { Console.WriteLine ("Notificatore statico
                              \tavanzamento {0}%",a);
      return true; }
   public Logger(string id)
   { _id=id; }
}
```

vediamo che ci sono due metodi che stampano il progresso: uno statico e uno di istanza. Ecco come utilizzare il codice fin qui prodotto nella funzione *Main* della nostra applicazione di esempio:

```
public class Applicazione
{ static void Main(string[] args)
  { //creazione delle istanze
    ClasseCalcolo objCalcolo= new ClasseCalcolo();
    Logger objLog= new Logger("uno");
    //utilizzo del delegato di istanza
    objCalcolo.objDelegatoNotifica = new
         DelegatoNotifica (objLog.NotificaAvanzamento );
    objCalcolo.SimulaUnCalcoloLungo();
     //utilizzo del delegato col metodo statico
     objCalcolo.objDelegatoNotifica=null;
     objCalcolo.objDelegatoNotifica = new DelegatoNotifica(
                       Logger.NotificaAvanzamentoStatico);
          objCalcolo.SimulaUnCalcoloLungo(); }
}
```

Si vede dal codice come vengano utilizzati entrambi

i membri di *Logger*, statico e non, allo stesso modo: assegnando all'oggetto delegato *objDelegatoNotifica* un nuovo delegato dello stesso tipo inizializzato con l'identificatore senza parametri del metodo che vogliamo far impersonare dal nostro delegato *objDelegatoNotifica*. L'output del programma sarà:

```
Notificatore uno        avanzamento 0%
Notificatore uno        avanzamento 20%
Notificatore uno        avanzamento 40%
Notificatore uno        avanzamento 60%
Notificatore uno        avanzamento 80%
Notificatore statico    avanzamento 0%
Notificatore statico    avanzamento 20%
Notificatore statico    avanzamento 40%
Notificatore statico    avanzamento 60%
Notificatore statico    avanzamento 80%
```

L'oggetto delegato *objDelegatoFineCalcolo* non è stato chiamato in quanto non abbiamo assegnato a questo alcun metodo.

CATENE DI DELEGATI

Spesso abbiamo bisogno di compiere differenti azioni in risposta ad un evento. Nel nostro esempio potremmo aver bisogno di scrivere anche in un file di testo il procedere del nostro calcolo. Questo è possibile utilizzando il concatenamento dei delegati. *System.Delegate* espone un metodo statico, *Combine*, che ci permette di fare proprio questo. Ogni oggetto delegato è in realtà il nodo di una lista collegata che, per default, ha a *null* il puntatore al nodo precedente. Potremmo quindi far sì che venga definita una lista di delegati che verranno chiamati quando verrà invocato il nostro delegato multicast. In realtà, utilizzeremo l'operatore sovraccaricato + per aggiungere delegati alla lista. Vediamo subito il codice. *LogSuDisco* è la classe che scriverà i valori sul file di testo. Seguirà poi la versione aggiornata della funzione *Main* con il delegato multicast:

```
public class LogSuDisco
 {  string _fileName;
    public LogSuDisco(string fileName)
    { _fileName=fileName;}
    public bool Scrivi(int percentuale )
    { System.IO.StreamWriter SW= new
                System.IO.StreamWriter (_fileName,true);
      SW.WriteLine("{0}-calcolo al
                {1}%",DateTime.Now,percentuale);
      SW.Close();
      return true; }
    public bool ScriviFine(int risultato, DateTime ora)
    { System.IO.StreamWriter SW= new
                System.IO.StreamWriter (_fileName,true);
      SW.WriteLine("finito alle ore {0} con risultato
                {1}", ora, risultato);
      SW.Close();
      return true;} }
```

```
public class Applicazione
{ static void Main(string[] args)
  { ClasseCalcolo objCalcolo= new ClasseCalcolo();
    Logger objLog= new Logger("uno");
    LogSuDisco objLogSuDisco= new
                LogSuDisco("c:\\calcololog.txt");
    objCalcolo.objDelegatoNotifica = new
        DelegatoNotifica (objLog.NotificaAvanzamento );
    objCalcolo.objDelegatoNotifica += new DelegatoNotifica(
                Logger.NotificaAvanzamentoStatico);
    objCalcolo.objDelegatoNotifica += new
                DelegatoNotifica (objLogSuDisco.Scrivi );

   objCalcolo.objDelegatoFineCalcolo = new
        DelegatoFineCalcolo(objLogSuDisco.ScriviFine);
   objCalcolo.SimulaUnCalcoloLungo();  }
}
```

nel codice appena esposto vediamo come abbiamo aggiunto (+=) tre delegati ad *objDelegatoNotifica*. Quando questo verrà chiamato in *SimulaUnCalcoloLungo*, in realtà verranno invocati in sequenza i tre delegati aggiunti. Infatti oltre all'output sulla console, verrà anche prodotto un file di log. Si vede poi come è stato attivato anche il delegato *objDelegatoFineCalcolo*. Quindi ogni oggetto può registrare un suo metodo in uno o più dei delegati esposti, cosa non fattibile nella nostra prima soluzione con le interfacce. È interessante notare che l'output a video è ora differente da quello visto prima, in quanto i due metodi (statico e di istanza) vengono chiamati in sequenza per ogni evento:

Fig. 2: Utilizzando ILDASM si può vedere come il delegato sia una classe.

```
Notificatore uno        avanzamento 0%
Notificatore statico    avanzamento 0%
Notificatore uno        avanzamento 20%
Notificatore statico    avanzamento 20%
(...)
```

Un altro metodo per creare liste di delegati è quello di creare un array di delegati, passarlo al metodo statico *Delegate.Combine* e assegnare al delegato di catena il delegato risultante:

```
DelegatoNotifica[] delArray= new DelegatoNotifica[3];
delArray[0]= new   DelegatoNotifica(
                        objLog.NotificaAvanzamento);
delArray[1]= new   DelegatoNotifica(
                  Logger.NotificaAvanzamentoStatico);
delArray[2]= new   DelegatoNotifica(
                              objLogSuDisco.Scrivi);
```

```
objCalcolo.objDelegatoNotifica=(DelegatoNotifica)
                              Delegate.Combine ( delArray);
```

Dato che i delegati sono chiamati in sequenza, l'unico valore ritornato al chiamante è quello prodotto dall'ultimo metodo chiamato. A volte però potrebbe essere utile sapere il risultato di ogni chiamata: ci viene in aiuto un metodo esposto da *System.MulticastDelegate* (e quindi dai nostri oggetti delegati): *GetInvocationList()*, che restituisce un array di *Delegate* contenente i delegati presenti nella nostra lista:

```
public void SimulaUnCalcoloLungo()
{  int risultato=0;
   for (int a=0;a<100;a=a+20)
   { //if (objDelegatoNotifica!=null)
                                    objDelegatoNotifica(a);
     if (objDelegatoNotifica!=null)
     { foreach (DelegatoNotifica DN in
                    objDelegatoNotifica.GetInvocationList())
        { if (!DN(a))System.Diagnostics
                    .Debug.WriteLine ("Fallita la notifica!");}
     } risultato++;
     System.Threading.Thread.Sleep(1000); }
   if (objDelegatoFineCalcolo!=null)
   objDelegatoFineCalcolo(risultato, DateTime.Now ); }
```

possiamo così esaminare il risultato di ogni invocazione.

NOTA

FIRE AND FORGET

Una tecnica per l'utilizzo asincrono dei delegati è quella di non controllare il valore di ritorno, invocando il delegato con il codice:

```
mioDelegato.
BeginInvoke(parametri,
null, null)
```

DELEGATI ASINCRONI

Nella classe generata dal compilatore sono presenti altri due metodi che non abbiamo esaminato:

```
public virtual IAsyncResult BeginInvoke(int
    percentuale, AsyncCallback callback, object object);
public virtual bool EndInvoke(IAsyncResult result);
```

come è facile immaginare questi metodi permettono una gestione asincrona dei delegati. In questa sezione dell'articolo non ci addentreremo nei dettagli tecnici inerenti l'architettura multithreading sottostante le caratteristiche asincrone dei delegati, ma introdurremo alcune tecniche che possono essere utili. Il bisogno di una tecnica del genere può nascere da molte situazioni, la più comune delle quali è la necessità di riguadagnare il controllo dell'esecuzione della funzione chiamante senza attendere il ritorno della funzione chiamata. Si pensi a tutti quei casi in cui occorre far sì che l'interfaccia grafica dell'applicazione non sia bloccata, o si voglia chiamare in breve tempo tutti i delegati registrati. Se infatti una chiamata attraverso *Invoke()* (esplicito o implicito) non ritorna il controllo finché la lista dei delegati non ha finito l'esecuzione, una chiamata attraverso *BeginInvoke()* attiva la funzione chiamata ma ritorna subito il controllo. Occorre innanzitutto dire che non è possibile utilizzare in maniera asincrona delegati multicast a meno di non utilizzare la tecnica esposta nel paragrafo precedente attraverso l'utilizzo del metodo *GetInvocationList()*. Prima di esaminare tali tecniche è opportuno ricordare da quali proprietà è composta l'interfaccia *IAsyncResult*, che abbiamo visto nei metodi asincroni dei delegati:

```
public interface IAsyncResult
{ object AsyncState { get; }
  WaitHandle AsyncWaitHandle { get; }
  bool CompletedSynchronously { get; }
  bool IsCompleted { get; }
}
```

importante è la proprietà *IsCompleted*, che è *true* quando il metodo chiamato ha completato l'esecuzione. La prima tecnica per l'invocazione del metodo è molto semplice:

```
if (objDelegatoNotifica!=null)
{ foreach (DelegatoNotifica DN in
                    objDelegatoNotifica.GetInvocationList())
  { IAsyncResult AR= DN.BeginInvoke(a,null,null);
    while (!AR.IsCompleted )
    {  //fai qualcosa mentre aspetti!
      Sleep(1); }
    if (!DN.EndInvoke(AR) )System.Diagnostics.Debug
                          .WriteLine ("Fallita la notifica!");
  }
}
```

In questa tecnica il delegato viene chiamato in maniera asincrona, e poi viene eseguito un ciclo che controlla (*polling*) se l'esecuzione del delegato è completata. In questo caso chiamando *EndInvoke()* viene esaminato il risultato della chiamata. Questa tecnica di polling può essere utile per non bloccare l'interfaccia durante l'esecuzione dei metodi inglobati dai delegati. In alcuni casi è possibile che il metodo richiamato fallisca, o non ritorni in tempi brevi. In questo caso è possibile stabilire un intervallo di tempo entro il quale il metodo deve ritornare. Nell'esempio che segue si attende un timeout di sessanta secondi:

```
if (AR.AsyncWaitHandle.WaitOne(60000,true))
{ if (!DN.EndInvoke(AR) )System.Diagnostics.Debug.WriteLine
                               ("Fallita la notifica!"); }
else
{ Console.Writeline("Timeout!");}
```

Un metodo più potente per la chiamata asincrona è quello che utilizza un delegato di tipo

```
delegate void AsyncCallback(IAsynkResult ar)
```

vediamo un esempio. Ritornando al codice inizialmente scritto, possiamo invocare asincronamente un delegato passandogli una nuova istanza di un delegato:

```
(...)
objDelegatoNotifica.BeginInvoke(a, new AsyncCallback(
                FunzioneCallback), objDelegatoNotifica);
(...)
```

che fa riferimento ad un metodo del tipo:

```
void FunzioneCallback(IAsyncResult ar)
{ DelegatoNotifica DN=( DelegatoNotifica)ar.AsyncState;
  if(!DN.EndInvoke(ar) ) System.Diagnostics.Debug.WriteLine
                              ("Fallita la notifica!");}
```

Questa funzione viene chiamata una volta ritornato il metodo chiamato asincronamente. Questa tecnica permette di chiamare tutti i delegati in lista in tempi molto brevi, ed eventualmente utilizzare successivamente il valore di ritorno.

EVENTI

Gli eventi altro non sono che modificatori dei delegati, che aggiungono due accessor (*add* e *remove*) ai delegati (in realtà anche per gli eventi possiamo utilizzare +=). Quindi un evento protegge il delegato cui fa riferimento dall'accesso esterno, e permette l'uso della funzionalità di notifica tramite un meccanismo di pubblicazione/sottoscrizione. Inoltre un evento può far parte di una interfaccia. La dichiarazione dei delegati della nostra classe *ClasseCalcolo* diverrà dichiarazione di eventi:

```
public class ClasseCalcolo
{ public event DelegatoNotifica objDelegatoNotifica=null;
  public event DelegatoFineCalcolo
                        objDelegatoFineCalcolo =null;
(...) }
```

mentre per l'assegnazione dei delegati all'evento dovremo sempre utilizzare l'operatore +=:

```
static void Main(string[] args)
{ ClasseCalcolo objCalcolo= new ClasseCalcolo();
  Logger objLog= new Logger("uno");
  LogSuDisco objLogSuDisco= new LogSuDisco(
                              "c:\\calcololog.txt");
  objCalcolo.objDelegatoNotifica += new
      DelegatoNotifica (objLog.NotificaAvanzamento );
  objCalcolo.objDelegatoNotifica += new
    DelegatoNotifica(Logger.NotificaAvanzamentoStatico);
  objCalcolo.objDelegatoNotifica += new
             DelegatoNotifica (objLogSuDisco.Scrivi );
         objCalcolo.objDelegatoFineCalcolo += new
          DelegatoFineCalcolo(objLogSuDisco.ScriviFine);
  objCalcolo.SimulaUnCalcoloLungo(); }
```

se utilizzassimo la chiamata diretta:

```
objCalcolo.objDelegatoNotifica(20);
```

avremmo un errore. Tale chiamata sarebbe stata invece corretta nel caso in cui *objDelegatoNotifica* fosse stato un delegato e non un evento. Si noti che questa tecnica di pubblicazione/sottoscrizione per gli eventi è la stessa utilizzata dal framework per legare eventi WinForms o ASP.NET ai rispettivi gestori. A titolo di esempio si esamini il caso in cui sia stato aggiunto ad una form WinForms un pulsante, e sia stata associata una funzione all'evento di click sul tasto stesso. Visual Studio immetterà il seguente codice nel metodo di inizializzazione dei componenti (è omesso per chiarezza il codice di creazione del controllo):

```
private void InitializeComponent()
{  (...)
   this.button1.Click += new
          System.EventHandler(this.button1_Click);
   (...) }
```

ove viene passato all'evento *Click*, di tipo *System.-EventHandler*, il riferimento al metodo scelto per gestire la notifica di evento (in questo caso *button1_Click*). È opportuno notare che, sebbene quanto detto finora funzioni perfettamente, nelle linee guida di Microsoft per l'uso degli eventi si consiglia di utilizzare come parametri degli eventi l'oggetto sorgente e un parametro che contenga lo stato dell'evento stesso estendendo *System.EventArgs.* e di chiamare gli eventi con un nome che finisca per *EventHandler:*

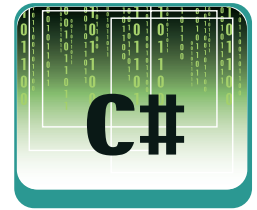
```
public delegate void MouseEventHandler(object
                         sender, MouseEvent e);
```

dove *MouseEvent* deriva da *System.EventArgs.*

CONCLUSIONI

Notificare e gestire eventi è alla base della programmazione moderna, sia per la logica di business che per la realizzazione delle interfacce grafiche. In questo articolo abbiamo visto come creare delegati e come utilizzarli in modo sincrono. Attraverso gli esempi forniti, e sbirciando "dietro le quinte" di quello che il compilatore C# combina con il nostro codice, abbiamo insomma fatto un po' di luce sul mondo interessante ma forse non semplicissimo dei delegati.

Marco Poponi

SUL WEB

Una divertente e completa introduzione ai delegati di Chris Sells:
http://www.sellsbrothers.com/writing/default.aspx?content=delegates.htm

Un bellissimo articolo di J. Richter sugli eventi
http://msdn.microsoft.com/msdnmag/issues/01/08/net/default.aspx

Le linee guida sugli eventi:
http://msdn.microsoft.com/library/en-us/cpgenref/html/cpconeventusageguidelines.asp

La definizione ECMA di C#
http://www.ecma-international.org/publications/standards/Ecma-334.htm

String matching

La comparazione tra stringhe è un compito di primaria importanza e di cui molte applicazioni, come elaboratori di testi o database management system, ne fanno un uso massiccio.

RICORSIONE
Particolare metodologia usata nella programmazione per la quale una funzione richiama se stessa. Si riscontra ricorsione anche quando una funzione *A* richiama un'altra funzione *B* che a sua volta richiama *A*; comunque quando anche dopo una catena di chiamate a funzioni si richiama una delle routine in cima all'albero delle chiamate. Un semplice esempio di ricorsione è la funzione che calcola il fattoriale, ecco come si presenta in forma ricorsiva.

```
function fatt(x:integer
                  ):integer;
begin
    if (x=0) or (x=1)
              then fatt:=1
    else fatt:=x*fatt(x-1);
end;
```

Va sottolineato che non tutti i linguaggi di programmazione supportano la ricorsione.

Un po' per gioco e un po' seriamente più volte abbiamo passato la nostra lente di ingrandimento sulla semplice, quanto ricca di contenuto informativo, entità quale è la parola. Dopo aver ricercato anagrammi, esplorato tecniche di clustering per il raggruppamento di testi ed esaminato metodi come il *t9*, è arrivato il momento di confrontare singole parole. Tratteremo più precisamente di stringhe e di come tali sequenze alfanumeriche si possano raffrontare in modo da estrarre elementi che indichino il grado di "similitudine". È innegabile l'importanza cruciale di tali algoritmi. Si pensi ai moderni correttori ortografici che oltre a rilevare errori di non corrispondenza al dizionario propongono, correzioni automatiche, ossia individuano la parola o le parole che più "somigliano" a quella errata. Il comando *like* presente in alcuni linguaggi e *DBMS* permette di confrontare stringhe che siano "simili". Insomma, vi sono tanti esempi di campi di applicazione nella programmazione per il confronto di parole, quello che in letteratura informatica è conosciuto come *string matching*. Purtroppo l'argomento non si riduce al semplice confronto tra due stringhe, ma si caratterizza di metodi avanzati per implementare, al meglio, la comparazione. Il problema che si presenta davanti al programmatore è attuare le azioni che precedentemente erano virgolettate; si vuole, in altri termini, formalizzare il concetto di similitudine e successivamente risolverlo. Lo stato dell'arte non può ritenersi concluso, infatti, si riscontrano sempre nuovi contributi atti a rendere le soluzioni più efficienti. Nel nostro studio esamineremo le tappe fondamentali che conducono all'obiettivo preposto, svilupperemo il codice relativo in pascal.

UN PRIMO APPROCCIO

Una discriminante che classifica gli algoritmi che ci apprestiamo a focalizzare è l'obiettivo che vogliamo perseguire. A seconda delle applicazioni si potrebbero avere diverse esigenze. In alcune semplici casi si vuole soltanto verificare se una data stringa è contenuta in un'altra, in altre ciò non è più sufficiente. Come vedremo tra poche righe tale problema è molto più semplice di altri, sia in fase di implementazione ma soprattutto nella formalizzazione. Per cui per "riscaldarci" esamineremo un algoritmo "ingenuo" che verifica l'inclusione di una parola in un'altra, successivamente ci attrezzeremo per definire metodi che possano descrivere il concetto di similitudine tra stringhe e quindi svilupparli. Il pascal oltre che, per la naturale propensione alla chiarezza, è l'ideale all'uopo per l'ottima gestione delle stringhe. Esaminiamo la funzione e rimandiamo un'analisi dei dati più approfondita per le successive funzioni. Considerate due stringhe (contraddistinte dai due parametri formali *s1* e *s2*) e le relative lunghezze ottenute con la funzione *length*. Per non incappare in errori di compilazione la seconda delle stringhe deve avere sempre lunghezza minore. All'interno del ciclo semplicemente si verifica se i caratteri correnti delle due stringhe sono uguali, come si può notare i due indici che scorrono su di essi sono rispettivamente *i* e *j*. Qualora tale eguaglianza sia verificata, entrambi gli indici scorrono in avanti, ossia vengono incrementati, altrimenti vengono nuovamente inizializzati. Ciò significa che l'indice *j* della seconda stringa, conosciuta come target, deve essere azzerato; il che equivale all'azione di riesaminare nuovamente il target. Per la prima stringa, invece, l'indice *(i)* deve ripartire dal valore immediatamente successivo al corrente decrementato della lunghezza di porzione di stringa risultata uguale al passo precedente, che è appunto il valore *i-j+1*. Se la stringa *s2* è inclusa, la funzione restituisce il valore booleano *true*.

```
function incluso(s1,s2:string):boolean;
var l1,l2,i,j:integer;
Begin
    l1:= length(s1);
    l2:= length(s2);
    writeln(l1,'  #   ',l2);
    i:=0;  j:=0;
    repeat
      i:=i+1; j:=j+1;
      if (s1[i] <> s2[j]) then
      begin
          i:=i-j+1;
```

```
            j:=0;
        end;
        until  ((j=l2) or (i=(l1-l2+1)));
        if (j=l2) then writeln('La stringa e'' stata trovata
                                          in posizione ',i-l2)
        else writeln('La stringa non e'' stata trovata');
      incluso:=(j=l2);
End;
```

Lo strumento è sovente, presente in molti editor, può essere efficacemente implementato con la routine codificata.

VERSO SOLUZIONI PIÙ EFFICIENTI

Mentre per alcune tipologie di problemi sono sufficienti soluzioni simili alla precedente, le stesse risultano inadatte ad altre esigenze. Quando si vuole tentare il matching che realizzi la similarità bisogna percorrere altre strade. Proporremo una soluzione che affronta il problema nella sua globalità e svilupperemo diverse versioni sempre più efficienti. Va detto, comunque, che esistono altri metodi dalle migliori performance che saranno oggetto delle nostre analisi nella prossima puntata. Si vuole individuare la massima sottosequenza comune. Come vedremo si tratta di un metodo che confronta due o più stringhe tentando di individuare similarità tra esse. Ma prima di entrare nel merito è necessario dare alcune definizioni. Si definisce *sottosequenza una stringa y di una stringa x se y è ottenibile da x eliminando dei caratteri di x.* Il numero di caratteri potrebbe fornire un indice di similarità che comunque non calcoleremo. È ovvio che, se il numero di caratteri da cancellare, è zero si ha un elevata similarità (nel caso specifico è proprio eguaglianza), bassa similarità in situazione diametralmente opposta, con un elevato numero di cancellazioni. Un ulteriore problema che sorgerebbe, è stabilire quanto un numero è elevato per considerare poca similarità tra stringhe ma anche questo esula dai nostri scopi. Si rende necessaria una definizione formale. *Una stringa y1, y2,… yi è sottosequenza di un'altra x1, x2,… xm qualora esista una sequenza di interi strettamente crescente z1, z2,…zi con 0<zi<m tale che yi=xzi.* D'accordo, la prima definizione è più intuitiva! La seconda ci servirà per sviluppare il programma e ha il dono della formalità. Quando si confrontano delle stringhe però, il numero di sottosequenze comuni può essere elevato, inoltre, quelle che hanno una bassa similarità generalmente non ci interessano, ecco che si estrae tra esse quella o quelle che massimizzino il numero di elementi della stringa. Una massima sottosequenza comune tra *n* stringhe *S1, S2… Sn* è quella o quelle con il maggior numero di caratteri. Non necessariamente una sola stringa ha questa caratteristiche, possono averla più di una. È interessante che vengano coinvolte più parole e che per esse si ricerchi gli elementi comuni, visti come massima sottosequenza comune. Inoltre, come da pregressa specifica si tende ad individuare la similarità, infatti, si tiene conto sia della eguaglianza tra caratteri, ma anche del dell'ordine degli stessi che deve essere rispettato. Giusto un inciso su questo ultimo concetto. Se si tenesse conto della sola eguaglianza tra caratteri, risulterebbero simili anche semplici anagrammi, il che non centra le nostre aspettative sul tema. In sintonia con le funzioni che si stanno implementando indicheremo con *MaxSubSequ* la massima sequenza comune mentre con *MaxSubSequVal* la cardinalità (lunghezza) di tale stringa. Vediamo alcuni esempi per chiarire.

```
MaxSubSequVal(paglia,piga) vale sia pia che pga
MaxSubSequVal(algoritmo,logaritmo) è laritmo
MaxSubSequVal(papavero,papero) è papero
MaxSubSequ(paglia,piga)=3
MaxSubSequ(algoritmo,logaritmo)=7
MaxSubSequ(papavero,papero)=7
```

Si ribadisce, come in alcuni casi, la soluzione non sia unica, anche se come vedremo l'algoritmo ne genera una sola. Ad esempio, le due parole *paglia* e *piga* generano due massime sottosequenze comuni, ossia *pia* e *pga*. Il nome della funzione è autoesplicativo, *val* sta per valore e indica, al contrario della successiva funzione che ha come output un numero, che si produce una stringa.

LE IMPLEMENTAZIONI

Dedicheremo la nostra attenzione dapprima al caso di valutazione di due singole stringhe. La generalizzazione non è un compito particolarmente ostico. Il primo algoritmo che svilupperemo calcola la lunghezza della massima sequenza comune (il secondo blocco di funzioni nell'esempio appena accennato). A partire da questo, troveremo l'effettiva stringa. Una prima formulazione naturale dell'algoritmo è di natura ricorsiva. Alla funzione porremo il suffisso *ric,* per distinguerla dalla successiva che, come vedremo, ha uno sviluppo iterativo.

Le due stringhe indicate con *s1* e *s2* possono trovarsi in due situazioni. Il primo è che il carattere finale per entrambe sia identico, in tal caso bisogna riapplicare il metodo alle sottostringhe *s1* ed *s2* private dell'ultimo carattere e incrementare il contatore per il calcolo del numero ricercato. Si deve cioè fare *MaxSubSequRic:=MaxSubSequRic(s1,s2,i-1,j-1)+1;* con *i* e *j* indici correnti delle due stringhe. Il secondo caso si ha quando, non essendo verificata la condizione precedente, bisogna calcolare *MaxSubSequRic* percorrendo due possibili strade. Si valutano,

Il trattamento automatico delle parole è stato proposto diverse volte su questa sezione della rivista. Ricordo alcuni miei articoli che hanno significativi legami con il presente. Nel numero 74 sono stati trattati i metodi predittivi di composizione dei testi, come il t9, usato per telefoni cellulari. I numeri 65 e 66 hanno affrontato l'affascinante mondo dell'anagrammatica. Infine, rispolverando il mio DB viene segnalato un articolo sul confronto tra stringhe al numero 17, si tratta di un pezzo dello scorso millennio!

infatti, due coppie di stringhe. La prima coppia è *s1* ed *s2* privata dell'ultimo carattere; la seconda è *s1* privata dell'ultimo carattere e *s2*. Tra le due coppie si sceglie quella che presenta coefficiente (output) maggiore. In entrambi i casi, come si nota, vi è una valutazione di *MaxSubSequRic* per stringhe via via più corte; è evidente, quindi, la strutturazione ricorsiva. Il criterio di uscita dalla funzione che da inizio alle pop sullo stack delle chiamate, ossia il percorso a ritroso delle funzioni chiamanti è il caso in cui una delle stringhe sia vuota.

```
function MaxSubSequRic(s1,s2:string; i,j:integer
                                          ):integer;
begin
    if (i<=0) or (j<=0) then MaxSubSequRic:=0
    else if s1[i]=s2[j] then MaxSubSequRic:=
                          MaxSubSequRic(s1,s2,i-1,j-1)+1
        else MaxSubSequRic:=max(MaxSubSequRic(
                  s1,s2,i-1,j),MaxSubSequRic(s1,s2,i,j-1))
end;
```

Per quanto sia naturale la scrittura di tale codice tanto è inefficiente. Ad ogni generica chiamata si richiamano due funzioni applicate a due coppie di stringhe che, a loro volta, nel caso generico, devono richiamare altre due volte la funzione. La continua biforcazione genera un albero di chiamate che vede circa *2n nodi,* nel caso in cui le due lunghezze di stringhe siano comparabili. Il punto debole di tale algoritmo è che, per calcolare il valore di massima sequenza comune tra due stringhe, bisogna valutare tutti i valori delle stringhe man mano più corte. Tale mansione bisogna ripeterla ad ogni passo. La cosa è ancora più pesante se si pensa che il calcolo ogni volta genera un albero di chiamate. La soluzione diventa semplice mediante la predisposizione di una struttura dati che memorizzi di volta in volta la lunghezza delle massime sottosequenze comuni. Si presta agevolmente al compito una matrice che abbia dimensioni di riga e colonna rispettivamente pari alla lunghezze delle due stringhe. L'elemento generico *(i,j)* indica, fino a quel momento, quale sia la lunghezza della massima sottosequenza comune. La formulazione si trasforma in iterativa con l'introduzione di due cicli innestati che rendono possibile la scansione per ogni carattere di una stringa di tutti i caratteri della seconda stringa.
Esaminiamo il codice. Le due funzioni di servizio, *inizializza* e *max,* che rispettivamente predispongono la matrice portando a zero i valori della prima riga e della prima colonna, e che calcolano il massimo tra due numeri interi, sono riportate per completezza nel prossimo paragrafo.

```
function MaxSubSequIt(s1,s2:string; var mat:matrice
                                          ):integer;
var   i,j:integer;
begin
    inizializza(mat,length(s1),length(s2));
    for i:=1 to length(s1) do
        for j:=1 to length(s2) do
            if s1[i]=s2[j] then mat[i,j]:=mat[i-1,j-1]+1
            else mat[i,j]:=max(mat[i-1,j],mat[i,j-1]);
    MaxSubSequIt:=mat[length(s1),length(s2)]
end;
```

Quando si incontra un'eguaglianza tra caratteri si incrementa il valore corrente della lunghezza del massimo valore che si riscontrava fino a quel momento che è indicato in posizione *(i-1,j-1)*, altrimenti si assegna, semplicemente, il valore massimo tra gli elementi adiacenti in posizione *(i,j-1)* e *(i-1,j)*. Al termine *MaxSubSequIt* sarà l'ultimo valore (vertice in basso a destra indicato nell'esempio in rosso) della matrice.
È interessante e istruttivo analizzare la matrice prodotta per un caso specifico.

		P	I	G	A
	0	0	0	0	0
P	0	1	1	1	1
A	0	1	1	1	2
G	0	1	1	2	2
L	0	1	1	2	2
I	0	1	2	2	2
A	0	1	2	2	3

TABELLA 1: Matrice per il calcolo della lunghezza della massima sottosequenza comune tra due stringhe. L'output prodotto è 3.

Si deve tenere presente che la reale matrice è quella riportata con doppia bordatura. Gli elementi esterni aiutano a capire come l'array viene costruito. Gli indici partono da zero e arrivano alle lunghezze delle due stringhe (ottenibili come specificato nel codice dalle funzioni predefinite *length*).
La prima riga e la prima colonna servono da inizializzazione, infatti, come specificato, si fa riferimento a elementi del tipo *i-1* che, nella circostanza, servono alle prime iterazioni del ciclo. Per terminare siamo interessati a ottenere l'effettiva stringa comune. Utilizzando come elemento di input la matrice appena prodotta si ottiene un efficiente soluzione con formulazione iterativa.
Bisogna scandire la matrice a ritroso. Percorrendo i punti in cui vi sono dei caratteri in comune. Se le due stringhe presentano caratteri uguali allora bisogna decrementare entrambi gli indici di scansione.
Per comprendere nei dettagli si tenga sottocchio la matrice di esempio precedentemente ottenuta. Se invece, non c'è eguaglianza, bisogna risalire per riga o colonna a seconda del valore maggiore che si riscontra in tali posizioni. La stringa temporanea *temp*, vuota inizialmente, concatena, di volta in volta, i caratteri uguali osservati; al termine produrrà l'output.

```
function MaxSubSequVal2(s1,s2:string;
                                mat:matrice):string;
var temp : string;
   i,j : integer;
begin
   temp:='';
   i:=length(s1);  j:=length(s2);
   while (i>0) and (j>0) do
   if s1[i]=s2[j] then
   begin
      temp:=s1[i]+temp;
      i:=i-1; j:=j-1;
   end
   else if mat[i-1,j]>mat[i,j-1] then i:=i-1
      else j:=j-1;
   MaxSubSequVal2:=temp;
end;
```

Con riferimento all'esempio precedente, con input le due parole *paglia* e *piga* la funzione individua delle due possibili massime sottosequenze comuni *pia.*

PER COMPLETARE IL CODICE

Per completezza, di seguito è riportato il codice che manca per rendere l'intero progetto (programma) eseguibile. Sono presenti, oltre che l'intestazione e la sezione dichiarativa, anche le funzioni di servizio *max, inizializza* e *visual*. Per terminare vi è un possibile main program che richiama tutte le routine sviluppate, il che consente una visione di tutto lo sviluppo. La dimensione massima è stata settata a 100. Le variabili globali sono *parola_1* e *parola_2* identificano gli oggetti delle nostre elaborazioni. La matrice analizzata è *mssp* (*massima sottosequenza parziale*).

```
Program string_matching;
const DIM=100;
type matrice = array[0..DIM,0..DIM] of integer;
var parola_1,parola_2 : string;
          mssp : matrice;
function max (x,y:integer): integer;
begin
   if x>y then max:=x
   else max:=y
end;
procedure inizializza(var mat:matrice; l1,l2:integer);
var i,j:integer;
Begin
   for i:=0 to l1 do mat[i,0]:=0;
   for j:=0 to l2 do mat[0,j]:=0;
End;
procedure visual(mat:matrice; l1,l2:integer);
var i,j:integer;
Begin
   for i:=0 to l1 do
   Begin
      for j:=0 to l2 do
         write(mat[i,j]:4);
         writeln;
   End
End;
...... (* Altre funzioni e procedure *)
begin (* Principale *)
   write('inserisci la prima parola : ');
   readln(parola_1);
   write('inserisci la seconda parola : ');
   readln(parola_2);
   if incluso(parola_1,parola_2) then writeln(
                     parola_2,' e'' incluso in ',parola_1)
   else writeln(parola_2,' non e'' incluso in ', parola_1);
   writeln(MaxSubSequRic(parola_1,parola_2,length(
                        parola_1),length(parola_2)));
   writeln(MaxSubSequIt(parola_1,parola_2,mssp));
   visual(mssp,length(parola_1),length(parola_2));
   writeln(MaxSubSequVal2(parola_1,parola_2,mssp));
   readln;
End.
```

Chi volesse ricostruire l'intero programma deve riferirsi a questo scheletro, e aggiungere le funzioni che abbiamo prodotto negli altri paragrafi, ovviamente, prima del main program; oppure semplicemente può consultare il codice presente su CD e/o sul Web. In questa versione sono richiamate, in sequenza, tutte le routine sviluppate, per renderlo più user friendly si può prevedere un menu di scelta.
Generalizzare al caso di tre stringhe significa predisporre tre cicli innestati e costruire una matrice a tre dimensioni; ogni spigolo è una parola; quindi il procedimento è analogo al presente.
Quattro stringhe: quattro cicli e matrice a quattro dimensioni e così via. È stata attuata una generalizzazione che implementa *n* cicli innestati con l'uso di ulteriori strutture dati. Ecco uno spunto. Si provi a generalizzare l'algoritmo.

CONCLUSIONI

Le soluzioni esaminate sono un piccolo sottoinsieme delle conosciute. La letteratura presenta molte teorie, solo per citarne qualcuna esistono quella di Karp Rabin dei primi anni 80, successivamente raffinate da studi di Knuth, Morris e Pratt; ma hanno una significativa valenza le tecniche che fanno uso di automi. Molto affascinanti infine quelle che si fondano sulle reti neurali.
Nel prossimo appuntamento ci soffermeremo su alcuni di questi approcci più complessi e di maggiore efficienza.
Non resta quindi che attendere il prossimo numero.

Fabio Grimaldi

Alla ricerca della combinazione esatta

N loop innestati

di Fabio Grimaldi

Generalizzare n cicli è la chiave che conduce alla soluzione per il problema dell'individuazione delle combinazioni semplici di *n* elementi raggruppati in *k* per volta. Una tecnica utile anche in diverse altre circostanze.

Ricordo che una delle due sfide lanciate nella scorsa puntata richiedeva di individuare un metodo automatico per la ricerca di combinazioni semplici di n elementi raggruppati a classi di *k*. Attenzione perché la richiesta non si limitava al semplice calcolo del numero di tali combinazioni per il quale basta applicare la formula:

$$C_{n,k} = \frac{n}{kò*(n-k)!}$$

Si vogliono comporre tutte le combinazioni di n oggetti a gruppi di *k*. Con riferimento all'esempio dell'urna si vuole rispondere ad una domanda del tipo: Quali sono tutte le possibili combinazioni di n palline (distinguibili tra loro) a gruppi di *k?*

LA SOLUZIONE

Inizieremo con un caso semplice, per il quale produrremo una soluzione riferita al problema specifico. Successivamente generalizzeremo. Supponiamo di possedere una urna con cinque palline contrassegnate dalle prime cinque lettere dell'alfabeto. E di voler calcolare tutti i gruppi di due. Improntiamo una soluzione. Definiamo dapprima una struttura dati adeguata. In un vettore di caratteri di dimensione cinque *(n)* introdurremo gli elementi, rappresentanti le cinque palline.

v

a	b	c	d	e

Per formare le combinazioni di due elementi è conveniente adottare il seguente procedimento. Si abbina il primo elemento del vettore con tutti gli altri in modo da ottenere:

ab, ac, ad, ae

Poi, si considera il secondo elemento e lo si associa a tutti i caratteri alla sua destra (si tenga presente l'ordine in cui appaiono le palline nel vettore), giacché la combinazione che si otterrebbe dal raggruppamento con l'elemento a sinistra *(ba)* è stata già ottenuta. Si ricorda che l'ordine nelle combinazioni non è elemento distintivo tra gruppi, lo è per le disposizioni. Dopo questa seconda fase, che come vedremo corrisponde alla seconda esecuzione di un loop si ottengono le altre combinazioni:

bc, bd, be

Proseguendo con questa tecnica si definiscono altre tre configurazioni:

ed, ce
de

In totale le combinazioni sono dieci, come si può verificare dall'applicazione della formula. Ma implementiamo quello che abbiamo scritto in modo informale, per farlo utilizziamo un linguaggio dalla sintassi chiara e semplice, il Pascal.

```
for i := 1 to n do
  for j := i+1 to n do
    write(v[i]:4,v[j]:1);
```

Per il codice nella sua completezza si rimanda al prossimo paragrafo dove è proposto l'intero programma. La procedura appena sviluppata è *combina_2*. Con i due indici *i* e *j* si accede ai caratteri di ogni combinazione. Il secondo ciclo su *j* ci assicura che vengano raccolti elementi a destra del primo (indicato con *i*) grazie al fatto che si parte da i+1. Qualora *k* aumenti di uno, il metodo appena esposto si rivela inadeguato. Per *k* pari a tre è necessario un altro indice, quindi un altro ciclo. In analogia al metodo appena esaminato si scrive:

```
for i := 1 to n do
 for j := i+1 to n do
   for h := j+1 to n do
     write(v[i]:4,v[j];1,v[k]:1)
```

Si veda la procedura *combina_3*. Purtroppo non abbiamo centrato ancora il nostro obiettivo visto che intendiamo estrarre un metodo generale che valga per qualsiasi *n* e qualsiasi *k* (salvo limiti della macchina). La soluzione è dietro l'angolo ma non è banale. Bisogna generalizzare gli *n* cicli innestati. Abbiamo, infatti visto, che con due cicli risolviamo il problema per *k=2,* con tre risolviamo per *k=3,* con quattro per *k=4* e così via. Per pervenire alla soluzione dobbiamo utilizzare delle strutture dati aggiuntive, al bisogno si prestano due vettori che indicheremo rispettivamente con *c* e *l*. Entrambi i vettori hanno ampiezza *k*.

c

1	2	3

l

3	4	5

Il vettore *l* è quello dei limiti, cioè i valori a cui devono arrivare *i* contatori che sono in *c*. Il vettore *c* indica la combinazione corrente. Ad esempio, al primo passo, si ha il raggruppamento i cui elementi del vettore *v* sono quelli di indice *1, 2* e *3* (gli indici sono i contenuti delle celle dell'array *c*) ovvero *v[c[1]], v[c[2]]* e *v[c[3]]*. Utilizzando il vettore c non sono necessarie *k* variabili contatore ma è sufficiente un array di lunghezza *k*. Ma soprattutto non si devono implementare loop innestati: basta un unico ciclo, si faccia riferimento alla procedura *combina_k*. Le prime righe indicano le operazioni di preparazione con le quali si legge *k* ed il vettore *v* e si inizializzano i vettori *c* e *l* come spiegato sopra (in particolare *c*, nel caso *k=3*, dovrà essere inizializzato ai valori *1, 2, 2*, poiché nel programma principale vi è prima l'incremento di *c* e poi il suo uso; tale ragionamento vale ovviamente anche per gli altri valori di *k*). Con il solo loop esterno si simulano tutti i cicli annidati, facendo uso della struttura di supporto array *c*. Il *for* ci permette di scorrere sul vettore *c* e quindi di visualizzare la configurazione corrente, che ogni qual volta si entra nel ciclo viene rinnovata. È la procedure *incrementa* che si occupa di tale questione. Si tenta sempre di aggiornare la cella più a destra. Ad esempio, l'aggiornamento della configurazione

alla seconda combinazione è

c

1	2	4

Qualora la cella più a destra abbia raggiunto il suo valore massimo, cioè se è pari al valore corrispondente del vettore *l*, si proverà con il valore immediatamente a sinistra così fino ad arrivare alla prima cella di *c*. Se i valori delle prime celle sono uguali *(c[1]=l[1])* allora vorrà dire che sono state esaminate tutte le combinazioni. Se il numero aggiornato non è quello in fondo a destra si renderà necessario l'incremento del valore corrente della cella del vettore *c* e tutti gli elementi a destra di essa. Tale operazione è svolta nella procedura incrementa dal ciclo *for*; è il caso in cui ad esempio, si vuole passare dalla combinazione:

Alla configurazione

Come si può notare si incrementa il secondo elemento e si aggiorna il terzo. Una variabile booleana *incr* ci dice se si sono individuate nuove combinazioni, tale variabile verrà quindi usata per uscire dal programma.

IL CODICE

Per completezza è riportato l'intero programma compilabile. Le parti salienti sono gia state discusse in analisi. Non sono state esposte le routine di servizio che si occupano di compiti come la lettura dell'urna (vettore), procedure *carica*, l'inizializzazione dei vettori *l* e *c* prodotta nella procedura *inizializza* (si faccia attenzione alla distinzione tra i parametri formali *pl* e *pc* usati all'interno della routine e i parametri attuali *l* e *c*) ed infine, il main program dove vengono richiamate in sequenza tutte le procedure per il calcolo delle combinazioni.

```
Program combinazioni;
const DIM=10;
type vettoreogg = array[1..DIM] of char;
   vettore = array[1..DIM] of integer;
var oggetti : vettoreogg;
   n : integer; (* dimensione vettore(numero oggetti) *)
(* lettura del vettore *)
Procedure carica (var v:vettoreogg);
var i:integer;
Begin
   write('numero oggetti -> ');
     readln(n);
     for i:=1 to n do
         Begin
             write('oggetto - ',i,' -> ');
             readln(v[i]);
         End;
End;
(* combinazioni di n elementi della classe 2*)
Procedure combina_2(v:vettoreogg);
var i,j:integer;
Begin
     writeln('Combinazioni di ',n,' della classe 2');
     for i := 1 to n do
     Begin
          for j := i+1 to n do
              write(v[i]:4,v[j]:1);
          writeln
     End;
     readln;
End;
(* combinazioni di n elementi della classe 3*)
Procedure combina_3(v:vettoreogg);
var i,j,h:integer;
Begin
     writeln('Combinazioni di ',n,' della classe 3');
     for i := 1 to n do
       for j := i+1 to n  do
          begin
             for h:=j+1 to n  do
                write(v[i]:4,v[j]:1,v[h]:1);
             writeln
          end;
     readln;
End;
(* Inizializzazione dei vettori di supporto c e l*)
procedure inizializza(var pc,pl:vettore; pk:integer);
var   i:integer;
Begin
     for i:=1 to pk do
     Begin
          pc[i]:=i;
          pl[pk-i+1]:=n-i+1
     End;
     pc[pk]:=pc[pk]-1
end;
```

N.1

Nel primo quesito eravamo chiamati a risolvere il problema di un ladro. Certo che è proprio un lavoro ingrato quello del programmatore, aiutare anche i malfattori. Il furfante su cento sacchetti di refurtiva (oro) apparentemente uguali, doveva distinguere l'unico più pesante avendo a disposizione una bilancia a due piatti e facendo il minor numero di pesate, per evitare che la polizia alle calcagna lo potesse fermare. L'idea migliore è dividere per tre l'intero malloppo. Anche se la divisione non è precisa, si pesano due gruppi da 33 sacchetti e ne rimane da parte uno da 34. Se il piatto della bilancia penderà da una delle due parti allora il sacchetto cercato è in quel gruppo, altrimenti se rimane in equilibrio vorrà dire che si trova nei rimanenti 34 sacchetti. Iterando il procedimento si divide ogni volta l'insieme di sacchetti per tre. Si scoprirà che il numero di pesate minimo è 5, che corrisponde alla risposta d. nessuna delle precedenti.

N.2

Per l'esercizio di programmazione l'analisi della procedura quack attraverso tabelle di verifica ci indica che il compito della routine è riempire *c* con gli elementi di *a* e *b* ordinati in modo crescente se gli elementi in *a* e *b* sono ordinati in modo crescente.

```
(* Individuazione di una nuova configurazione
                                         (combinazione) *)
Procedure incrementa(var pc:vettore; pl:vettore;
                                               pk:integer);
var cont1,cont2 : integer;
        incr : boolean;
Begin
        cont1:= pk; incr:=false;
    repeat
        if pc[cont1] < pl[cont1] then
        begin
            incr := true;
            pc[cont1] := pc[cont1]+1 ;
            for cont2:=cont1 to pk do
                pc[cont2+1] := pc[cont2]+1 ;
        end
        else  cont1 := cont1 -1;
    until (incr=true) or (pc[1]=pl[1])
End;
(* Combinazioni di n elementi della classe k *)
procedure combina_k(v:vettoreogg);
var  c,l : vettore;
    k,d : integer;
Begin
    Write('Classe k ->');
    readln(k);
    inizializza(c,l,k); // inizializzazione dei due vettori c e l
    writeln('Combinazioni di ',n,' della classe ',k);
    repeat
        incrementa(c,l,k);
        for d:=1 to k do
        write(v[c[d]]);// È presente un indicizzazione
                                                  indiretta
        writeln;
    until c[1]=l[1];          // condizione d'uscita
End;

Begin (* Programma Principale *)
    carica(oggetti);
    combina_2(oggetti);
    combina_3(oggetti);
    combina_k(oggetti);
    readln
End.
```

L'ANGOLO DELLA COMPETIZIONE

Per terminare proponiamo un nuovo problema. Questa volta esplorando l'archivio degli esercizi proposti nelle olimpiadi di informatiche passate, ne ho selezionato uno che ha una certa attinenza con gli argomenti trattati.

IL NEGOZIO DI FIORI

Si vuole allestire la vetrina di un negozio di fiori in un modo gradevole. Si dispone di *F* mazzi di fiori, ciascuno di un genere diverso, e almeno altrettanti vasi ordinati in una fila. I vasi sono incollati sopra a una mensola e sono numerati consecutivamente da *1* a *V*, dove *V* è il numero di vasi, disposti da sinistra a destra in modo che il vaso 1 è quello più a sinistra e il vaso *V* quello più a destra. I mazzi si possono spostare e sono identificati da un numero intero tra *1* e *F*. Questi numeri hanno un significato: determinano l'ordine richiesto di presentazione dei mazzi di fiori nella fila di vasi. Infatti, il mazzo *i* deve essere in un vaso alla sinistra di quello che contiene il mazzo *j* ogni qualvolta *i* < *j*. Supponiamo, per esempio, di avere un mazzo di azalee *(id=1)*, un mazzo di begonie *(id=2)* ed un mazzo di garofani *(id=3)*. Ora, tutti i mazzi devono essere messi nei vasi mantenendo l'ordine dei loro id. Il mazzo di azalee deve essere messo in un vaso alla sinistra delle begonie, e il mazzo di begonie in un vaso alla sinistra dei garofani. Se ci sono più vasi che mazzi di fiori allora i vasi eccedenti vengono lasciati vuoti. Un vaso può contenere solo un mazzo di fiori. Ciascun vaso ha una distinta caratteristica (esattamente come i fiori). Dunque, mettere un mazzo di fiori in un vaso ha come risultato un determinato valore estetico, espresso da un numero intero. I valori estetici sono mostrati nella tabella che segue.

		VASI				
		1	2	3	4	5
Mazzi	1 (azalee)	7	23	-5	-24	16
	2 (begonie)	5	21	-4	10	23
	3 (garofani)	-21	5	-4	-20	20

Lasciare un vaso vuoto ha un valore estetico di 0.
Secondo la tabella, l'azalea apparirebbe molto bene nel vaso 2, ma sarebbe esteticamente sgradevole nel vaso 4.
Raggiungere l'effetto più gradevole corrisponde alla massimizzazione della somma dei valori estetici per la sistemazione dei mazzi di fiori. Se più di una sistemazione ha il valore della somma massima, qualsiasi sistemazione tra queste sarà accettabile. Si deve produrre una sola sistemazione.

ASSUNZIONI

- **1 <= F <= 100, dove F è il numero dei mazzi di fiori. I mazzi sono numerati da 1 a F.**
- **F <= V <= 100, dove V è il numero di vasi.**
- **-50 <= Aij <= 50, dove Aij è il valore estetico ottenuto mettendo il mazzo i di fiori nel vaso j.**

DATI IN INPUT

L'input è un file di testo chiamato flower.inp.

- **La prima linea contiene 2 numeri: *F* e *V*.**
- **Seguono F linee, ciascuna contenente *V* numeri interi: Aij è dato come il j-esimo numero della (i+1)-esima linea del file di input.**

DATI IN OUTPUT

L'output deve essere un file di testo di nome flower.out che consiste in due linee:

- **La prima riga conterrà la somma dei valori estetici della sistemazione.**
- **La seconda riga dovrà presentare la sistemazione come un elenco di F numeri: il k-esimo numero su questa riga identifica il vaso nel quale il mazzo k è stato posto.**

Esempio di input e output

flower.inp:
- **3 5**
- **7 23 -5 -24 16**
- **5 21 -4 10 23**
- **-21 5 -4 -20 20**

flower.out:
- **2 4 5**

VALUTAZIONE

Al programma è concesso un tempo di esecuzione di 2 secondi.
Non sono previsti punteggi parziali.

KevLinDev

SVG, ossia Scalable Vector Graphics, è il nome di una nuova tecnologia destinata a far gola a quanti oggi sviluppano siti Web interattivi fortemente basati sulla comunicazione grafica.

C'è poco da fare: in questo momento la stragrande maggioranza dei filmati Web interattivi è realizzata sfruttando la tecnologia Flash di Macromedia. Il target di SVG, sostanzialmente, è lo stesso di Flash, ed i risultati che possono essere ottenuti con il primo sono pressappoco gli stessi che possono essere raggiunti dal secondo. Prevedibilmente, quindi, le due differenti tecnologie sono destinate ad entrare in competizione fra loro.

SVG CONTRO FLASH

Benché il fine di SVG e quello di Flash sia lo stesso, i mezzi usati per raggiungerlo sono profondamente differenti. Salvo imprevedibili colpi di scena futuri, analizzando quindi la situazione del momento, SVG sarà da preferirsi alla soluzione proposta da Macromedia. Anzitutto, mentre Flash rimane una tecnologia proprietaria, SVG è uno standard proposto e portato avanti da W3C, lo stesso consorzio che definisce numerosi altri standard per il Web, come HTML, XHTML, CSS, XML e così via. Dettagliate informazioni sullo standard SVG possono essere reperite direttamente sul sito Web di W3C (*http://www.w3.org/Graphics/SVG/*). In un futuro non troppo remoto, la tecnologia SVG sarà parte costituente di ogni browser moderno. Per eseguire filmati SVG, in parole povere, non sarà necessario installare un plug-in: il codice SVG potrà essere introdotto all'interno di una pagina Web esattamente come ora si incorporano fogli di stile CSS, porzioni di codice JavaScript o segmenti di MathML. Mozilla, il celebre browser open-source, sta sperimentando in questo momento il proprio supporto nativo ad SVG, ed è già possibile scaricare alcune build che ne comprendono una versione di prova. Finché SVG non sarà completamente integrato all'interno dei maggiori browser in circolazione, ad ogni modo, resta possibile appellarsi alla classica soluzione plug-in, proprio come per Flash. Giacché SVG è uno standard libero ed aperto, chiunque può realizzare plug-in per ogni tipo di browser. Tra i tanti che lo stanno facendo spicca il nome di Adobe, che ha intenzione di rendere SVG uno dei propri cavalli di battaglia (*http://www.adobe.com/svg/main.html*).

Fig. 1: *La Home Page di KevLinDev.*

Tralasciando le questioni di standardizzazione, SVG ha anche numerose altre caratteristiche vantaggiose. Ad esempio,

Fig. 2: *Uno degli esempi di KevLinDev, una riproduzione SVG del celebre gioco Asteroids.*

non soffre dei difetti di usabilità e accessibilità che da sempre sono il tallone di Achille di Flash. SVG è un formato testuale, basato su XML.

La programmazione degli eventi è affidata a JavaScript, mentre la definizione degli stili si basa sullo standard di CSS. Insomma, SVG è un coacervo di tecnologie e standard aperti, conosciuti e diffusi. Un più esauriente confronto tra i pregi ed i difetti di SVG e Flash è all'indirizzo *http://www.carto.net/papers/svg/comparison_flash_svg/*.

KEVLINDEV

KevLinDev, che sta per Kevin Lindsey Software Development, è uno tra i migliori siti che danno attualmente risalto alla tecnologia SVG. Installando il plug-in di Adobe sulla vostra macchina, o utilizzando qualsiasi altro visualizzatore, potrete saggiare "in anteprima" le possibilità e la potenza della nuova tecnologia. KevLinDev mostra SVG in azione attraverso numerosi esempi. Si va dal disegno e l'animazione di semplici forme geometriche fino ad applicazioni grafiche vettoriali 3D, passando per videogiochi ed esempi di interfacce grafiche. Ogni esempio mostrato ha forte valenza didattica: il sorgente SVG è sempre esplorabile e semplicemente comprensibile. Se state movendo i primi passi nel mondo di SVG, i tutorial presentati saranno certamente di vostro interesse. KevLinDev non copre, attualmente, tutto lo scibile di SVG, ma presenta elegantemente alcune ottime risorse. Coniungandole con le guide e le reference ufficiali presenti negli angoli Web di W3C e Adobe, potrete comprendere in pochissimo tempo i meccanismi di base di SVG e delle tecnologie collegate. Inoltre, KevLinDev è un sito ben fatto, usabile, accessibile e leggero. Ci si naviga che è un piacere.

Fig. 3: *Un altro esempio mostrato da KevLinDev, che combina la grafica vettoriale di SVG con il 3D.*

Carlo Pelliccia
carlo.pelliccia@ioprogrammo.it
L'autore ringrazia Marco Del Gobbo

INBox
L'esperto risponde...

Doppio clic su Java

Salute a tutti. Spero che mi possiate aiutare nel mio problemuccio: sono uno studente universitario e al momento sto studiando programmazione Java, i miei programmini girano a meraviglia sulla mia macchina in cui ho installato il jsdk 1.4.0-rc, ma come faccio a farli girare sui pc in cui è presente la sola java virtual machine (non sono presenti le librerie)? E poi come posso fare ad eseguirli con il classico doppio click sull'icona? In attesa di una vostra risposta vi saluto.

Elio

Risponde Paolo Perrotta

Per far girare un programma Java su una qualsiasi macchina, le librerie sono indispensabili quanto la macchina virtuale. Quello che non è invece indispensabile è il sistema di sviluppo *jsdk*, che include componenti aggiuntivi come il compilatore. Ti basta installare il Java Runtime Environment, che include le librerie, la VM e (sotto Windows) il plug-in per far girare le applet Java in Internet Explorer (il tutto può anche essere installato automaticamente via web se apri una pagina che include un'applet Java). Per far girare un programma con un doppio clic servono due cose:

1. devi compattare l'intero programma in un file JAR, che in pratica è uno ZIP che in più include un file che fornisce informazioni aggiuntive - ad esempio, dice qual è la classe che deve essere lanciata quando qualcuno apre il JAR (leggi la documentazione di Java per saperne di più)

2. i file .jar devono essere associati alla JVM (quest'associazione avviene spesso automaticamente quando si installa il Runtime - per verificarlo prova ad aprire la cartella DEMO del sistema di sviluppo e ad aprire uno dei file jar che ci trovi dentro - ad esempio, sul mio sistema, *C:\j2sdk1.4.2_02\demo\jfc\SwingSet2\SwingSet2.jar*).

Trasferire i dati da un foglio excel a una tabella in oracle

Devo trasferire i dati presenti in un foglio Excel in una tabella di oracle, sapete come fare?

m. vasquez

Risponde Elia Florio

Salva il foglio Excel come file CSV (comma separed value, campi separati da **;** o da **,**) e poi fatti caricare il file CSV dentro Oracle usando SQLLoader (SQLLDR.EXE) e un file .CTL scritto da te che spiega ad Oracle come importare i campi. Cerca un pò di esempi sull'uso di SQLloader e su come creare un .CTL. È il modo più veloce per caricare migliaia di dati in Oracle.

Passare array ad una function

Dovrei passare array ad alcune function per migliorare la leggibilità del codice però non so come fare. Qualcuno mi può aiutare? Grazie

fedeDev

Risponde infomaster

Basta usare la parola chiave ParamArray, mi spiego con un esempio:

```
'Somma i numeri in ingresso
Private Function Somma(ParamArray Valore
                        as Variant) as Long
Dim x as Integer
For x = LBound(Valore) to UBound(Valore)
--Somma = Somma + Valore(x)
Next x
End Funtion
'richiamo la funzione
Private Sub CmdInit()
Dim Valori() As Integer,Indice as Integer
Dim Respo as VBA.VbMsgBoxResult
Do
--ReDim Preserve Valori(Indice)
--Valori(Indice)=InputBox("Inserisci il valore
                  che vuoi" & ----" sommare:",
                  "INSERIMENTO GUIDATO",1)
--Respo = MsgBox("Vuoi inserire altri
                  valori?",VbYesNo)
Loop Until Respo = vbNo
LblRisultato.Caption = Somma(Valori)
End Sub
```

I/O sui file in C++

Allora...devo leggere da file di testo dei database di libri e utenti... la mia domanda e' questa: come faccio a dire al programma di leggere il file da un certo punto (ad ex dal carattere '#' che mi servira' per ricercare un libro per il suo codice ...#123456799#...) e a terminare ad un altro punto...?... e poi come faccio a scrivere alla fine del file la scheda di un nuovo libro/utente le cui caratteristiche (titolo,autore etc)saranno messe da me da input... aspetto risposte. Grazie ciao

The Prisoner

Risponde m.sturari

Ecco alcune funzioni per l'I/O: con questa funzione leggi un stringa dal file

```
char *fgets( char *string, int n, FILE
                                  *stream );
```

con questa funzione scrivi un stringa sul file

```
int fputs( const char *string, FILE *stream );
```

con questa funzione ti sposti all'interno del file

```
int fseek( FILE *stream, long offset, int
                                    origin );
```

con questa funzione leggi un blocco o un certo numero di blocchi di un certa dimensione

```
size_t fread( void *buffer, size_t size, size_t
                          count, FILE *stream );
```

con questa funzione scrivi un blocco o un certo numero di blocchi di un certa dimensione

```
size_t fwrite( const void *buffer, size_t size,
                   size_t count, FILE *stream );
```

Ci sono delle analoghe funzioni *_lseek*, *_lread*, *_lwrite* che usano l'handle al file al posto dello stream.
Tutto dipende dalla struttura che hai dato al file.

SXPER CONTATTARCI
e-mail: iopinbox@edmaster.it
Posta: Edizioni Master,
Via Cesare Correnti, 1 - 20123 Milano